

(19) 日本国特許庁 (J P)

## (12) 公表特許公報 (A)

(11) 特許出願公表番号  
特表2001-520415  
(P2001-520415A)

(43) 公表日 平成13年10月30日 (2001. 10. 30)

(51) Int.Cl. <sup>7</sup>	識別記号	F I	テマート* (参考)
G 0 6 F 9/45		G 0 6 F 9/34	3 5 0 A
9/34	3 5 0	9/38	3 5 0 A
9/38	3 5 0		3 8 0 B
	3 8 0	9/44	3 2 2 F

審査請求 未請求 予備審査請求 有 (全 65 頁)

(21) 出願番号	特願2000-516280(P2000-516280)	(71) 出願人	インスティテュート・フォー・ザ・ディベ ロップメント・オブ・エマージング・アー キテクチャーズ・エルエルシー アメリカ合衆国カリフォルニア州95014- 0795, クーパーチノ, ブルーンリッジ・ア ベニュー 19111, メール・ストップ 44 エル18, リーガル・デパートメント, ヒュ ーレット・パッカード・カンパニー内
(86) (22) 出願日	平成10年10月9日 (1998. 10. 9)	(72) 発明者	モリス, デイル・シー アメリカ合衆国カリフォルニア州94025, メンロ・パーク, ギルバート・アベニュー 442
(85) 翻訳文提出日	平成12年4月13日 (2000. 4. 13)	(74) 代理人	弁理士 社本 一夫 (外5名) 最終頁に続く
(86) 国際出願番号	P C T / U S 9 8 / 2 1 4 6 5		
(87) 国際公開番号	W O 9 9 / 1 9 7 9 5		
(87) 国際公開日	平成11年4月22日 (1999. 4. 22)		
(31) 優先権主張番号	0 8 / 9 5 3, 8 3 6		
(32) 優先日	平成9年10月13日 (1997. 10. 13)		
(33) 優先権主張国	米国 (U S)		
(31) 優先権主張番号	0 9 / 1 6 8, 0 4 0		
(32) 優先日	平成10年10月7日 (1998. 10. 7)		
(33) 優先権主張国	米国 (U S)		

(54) 【発明の名称】 命令実行を最適化する方法および装置

## (57) 【要約】

命令の投機実行中に遭遇した問題を先送りにする。投機的に実行した命令の結果をその後に用いる場合、命令実行の完全性を検証する。完全性が検証された場合、実行を継続する。検証されない場合、リカバリ・コードを実行し、コンピュータの状態を変更し、投機的に実行した命令の実行に成功したという状況を生じさせる。一実施形態では、投機的に実行し例外状態に遭遇した命令はいずれも延期例外トークンを発生する。延期例外トークンは、投機的命令全体を伝搬する。投機チェック命令が、延期例外トークンの存在をチェックする。トークンが見つからない場合、命令の投機的実行の完全性が検証されたことになり、通常に実行を継続する。トークンが見つかった場合、投機チェック命令がリカバリ・コードを活性化して、非投機的に命令を再実行することにより、例外を再現する。別の実施形態では、ロードおよびこれに依存する計算命令は、ロードと衝突する可能性のある格納よりも前に繰り上げられる。メモリ・チェック命令を実行し、メモリ・アクセスが実際に独立していたか否かについて検証を行う。独立でなかった場合、メモリ・チ

ェック命令はリカバリ・コードを活性化し、メモリ・アクセスが適正な順序で実行されることを保証する。

**【特許請求の範囲】**

【請求項1】 コンピュータ読み取り可能な形態で格納されたコンパイルしたプログラムを有するコンピュータ読み取り可能媒体であって、前記プログラムが、

格納命令と、

前記格納命令の前にスケジュールされたロード命令と、

前記ロード命令によって読み出されるデータに依存する少なくとも1つの計算命令であって、前記格納命令よりも前にスケジュールされる、少なくとも1つの計算命令と、

前記格納命令および前記ロード命令がメモリ内の共通位置にアクセスするか否かについて判定を行うチェック命令と、

を備えるコンピュータ読み取り可能媒体。

【請求項2】 請求項1記載のコンピュータ読み取り可能媒体において、前記チェック命令が前記格納命令の後にスケジュールされている、コンピュータ読み取り可能媒体。

【請求項3】 請求項1記載のコンピュータ読み取り可能媒体において、前記格納命令が複数の格納命令を含む、コンピュータ読み取り可能媒体。

【請求項4】 請求項1記載のコンピュータ読み取り可能媒体において、前記格納命令および前記ロード命令が共通メモリ位置にアクセスする場合、前記チェック命令が制御フローを変更する、コンピュータ読み取り可能媒体。

【請求項5】 請求項2記載のコンピュータ読み取り可能媒体において、前記チェック命令が、アドレス・テーブルにアクセスし、前記格納命令および前記ロード命令がメモリ内の共通位置にアクセスするか否かについて判定を行い、前記格納命令の実行後、前記アドレス・テーブル内に前記ロード命令に対応するエントリがない場合、前記格納およびロード命令によってメモリ内の共通位置がアクセスされたことを示す、コンピュータ読み取り可能媒体。

【請求項6】 請求項4記載のコンピュータ読み取り可能媒体において、前記プログラムが、更に、前記格納命令および前記ロード命令が共通メモリ位置にアクセスする場合に制御を移すリカバリ・コードを含み、該リカバリ・コードが

、前記ロード命令および前記少なくとも1つの計算命令の再実行を含む、コンピュータ読み取り可能媒体。

【請求項7】 コンピュータ・システムであって、  
メモリと、  
格納命令を実行する手段と、  
前記格納命令の前にロード命令を実行する手段と、  
前記格納命令の前に、前記ロード命令によって読み出されたデータに依存する少なくとも1つの計算命令を実行する手段と、  
前記格納命令および前記ロード命令がメモリ内の共通位置にアクセスしたか否かについて判定を行う手段と、  
を備えるコンピュータ・システム。

【請求項8】 請求項7記載のコンピュータ・システムであって、更に、前記格納命令および前記ロード命令がメモリ内の共通位置にアクセスしたことを判定した場合リカバリ・コードに制御フローを変更する手段を含む、コンピュータ・システム。

【請求項9】 請求項7記載のコンピュータ・システムにおいて、前記格納命令が複数の格納命令を含む、コンピュータ・システム。

【請求項10】 請求項7記載のコンピュータ・システムにおいて、前記判定手段が、前記格納命令および前記ロード命令がメモリ内の共通位置にアクセスしたか否かを示す情報を含むテーブルを含む、コンピュータ・システム。

【請求項11】 請求項8記載のコンピュータ・システムにおいて、前記リカバリ・コードが、前記ロード命令および前記少なくとも1つの計算命令のコピーを含む、コンピュータ・システム。

【請求項12】 請求項8記載のコンピュータ・システムにおいて、前記判定手段が、前記格納命令および前記ロード命令がメモリ内の共通位置にアクセスしたか否かを示す情報を含むテーブルを含む、コンピュータ・システム。

【請求項13】 コンピュータ・システムであって、  
格納命令、ロード命令、および前記ロード命令によって読み出されるデータに依存する少なくとも1つの計算命令を含むソース・プログラムに対して実行スケ

ジュールを作成するコンパイラを備え、該コンパイラが、前記格納命令および前記ロード命令が前記プログラムの実行中共通メモリ位置にアクセスしないことに前記コンパイラが確信できない場合、前記ロード命令および前記少なくとも1つの計算命令を前記格納命令よりも前にスケジュールする手段を含む、コンピュータ・システム。

【請求項14】 請求項13記載のコンピュータ・システムにおいて、前記コンパイラが、前記プログラムの実行中、前記格納命令および前記ロード命令が共通メモリ位置にアクセスするか否かについて判定を行うチェック命令を発生する手段を含む、コンピュータ・システム。

【請求項15】 請求項14記載のコンピュータ・システムにおいて、前記スケジューラが、前記チェック命令を前記格納命令の後にスケジュールする手段を含む、コンピュータ・システム。

【請求項16】 請求項14記載のコンピュータ・システムにおいて、前記コンパイラがチェック命令を発生する手段を含み、該手段は、前記格納命令および前記ロード命令が共通メモリ位置にアクセスしたことを前記チェック命令が判定した場合、制御フローの変更を含む、コンピュータ・システム。

【請求項17】 請求項14記載のコンピュータ・システムであって、更に、複数の実行ユニットを含み、前記コンパイラが、更に、前記ロードおよび格納命令の実行を異なる実行ユニット上にスケジュールする手段を含む、コンピュータ・システム。

【請求項18】 請求項16記載のコンピュータ・システムにおいて、前記コンパイラが、更に、前記格納命令および前記ロード命令が共通メモリ位置にアクセスした場合に制御を移すりカバリ・コードを発生する手段を含み、該りカバリ・コードが、前記ロード命令および前記少なくとも1つの計算命令のコピーを含む、コンピュータ・システム。

【請求項19】 コンピュータ読み取り可能な形態で格納されたコンパイルしたプログラムを有するコンピュータ読み取り可能媒体であって、前記プログラムが、

格納命令と、

前記格納命令の前にスケジュールされたロード命令と、

前記プログラムの実行中、前記格納命令および前記ロード命令が共通メモリ位置にアクセスするか否かについて判定を行うチェック命令であって、該チェック命令が、前記格納命令および前記ロード命令が共通メモリ位置にアクセスしたと判定した場合、制御フローを変更する、チェック命令と、  
を備えるコンピュータ読み取り可能媒体。

【請求項20】 請求項19記載のコンピュータ読み取り可能媒体において、前記プログラムが、更に、前記プログラムの実行中前記格納命令および前記ロード命令が共通メモリ位置にアクセスしたと前記チェック命令が判定した場合に制御を移すリカバリ・コードを含み、該リカバリ・コードが前記ロード命令のコピーを含む、コンピュータ読み取り可能媒体。

【請求項21】 コンピュータ・システムであって、  
メモリと、

格納命令および以前に実行したロード命令が前記メモリ内の共通位置にアクセスしたか否かについてチェックする手段と、

前記格納命令および前記ロード命令が前記メモリ内の共通位置にアクセスしたと判定した場合、制御フローをリカバリ・コードに変更する手段と、  
を備えるコンピュータ・システム。

【請求項22】 請求項21記載のコンピュータ・システムにおいて、該コンピュータ・システムが、前記ロード命令のコピーを含む前記リカバリ・コードを発生する手段を含む、コンピュータ・システム。

【請求項23】 請求項21記載のコンピュータ・システムにおいて、前記チェック手段が、前記格納命令および前記ロード命令が前記メモリ内の共通位置にアクセスしたか否かを示す情報を含むテーブルを含む、コンピュータ・システム。

【請求項24】 コンピュータ・システム上で実行されたとき、格納命令と、ロード命令と、該ロード命令によって読み出されたデータに依存する少なくとも1つの計算命令とを含むソース・プログラムをコンパイルする方法を実行するコンパイラでエンコードしたコンピュータ読み取り可能媒体であって、前記方

法が、

(A) 前記プログラムの実行中、前記格納およびロード命令が共通メモリ位置にアクセスしないか否かについて判定を行うステップと、

(B) 前記格納およびロード命令が共通メモリ位置にアクセスしないと判定できない場合、前記ロード命令および前記少なくとも1つの計算命令を前記格納命令の前にスケジュールするステップと、  
を含むコンピュータ読み取り可能媒体。

【請求項25】 請求項24記載のコンピュータ読み取り可能媒体であって、前記方法が、更に、

(C) 前記プログラムの実行中、前記格納およびロード命令が共通メモリ位置にアクセスするか否かについて判定を行うチェック命令を発生するステップを含む、コンピュータ読み取り可能媒体。

【請求項26】 請求項25記載のコンピュータ読み取り可能媒体において、前記ステップ(C)が、前記チェック命令を前記格納命令の後にスケジュールするステップを含む、コンピュータ読み取り可能媒体。

【請求項27】 請求項25記載のコンピュータ読み取り可能媒体において、前記ステップ(C)が、前記格納命令および前記ロード命令が共通メモリ位置にアクセスしたと前記チェック命令が判定した場合、制御フローを変更するチェック命令を発生するステップを含む、コンピュータ読み取り可能媒体。

【請求項28】 請求項27記載のコンピュータ読み取り可能媒体において、前記方法が、更に、

(D) 前記チェック命令が、前記プログラムの実行中、前記格納命令および前記ロード命令が共通メモリ位置にアクセスしたと判定した場合に制御を移すリカバリ・コードを発生するステップを含み、該リカバリ・コードが、前記ロード命令および前記少なくとも1つの計算命令のコピーを含む、コンピュータ読み取り可能媒体。

【請求項29】 コンピュータ・システム上で実行されたとき、第1命令および第2命令を含むプログラムをコンパイルする方法を実行するコンパイラでエンコードしたコンピュータ読み取り可能媒体であって、前記コンパイラが、前記

第2命令が前記第1命令の実行に依存するデータ上で動作しないことを確信できない場合、前記方法が、

(A) 前記第2命令を前記第1命令の前にスケジュールするステップと、

(B) 前記プログラムの実行中、前記第1命令の実行に依存するデータ上で、前記第2命令が動作するか否かについて判定を行うチェック命令を発生するステップと、

を含むコンピュータ読み取り可能媒体。

【請求項30】 請求項29記載のコンピュータ読み取り可能媒体において、前記ステップ(B)は、前記第2命令が前記第1命令の実行に依存するデータ上で動作することを前記チェック命令が判定した場合に制御フローを変更することを含むチェック命令を発生するステップを含む、コンピュータ読み取り可能媒体。

【請求項31】 請求項30記載のコンピュータ読み取り可能媒体において、前記方法が、更に、

(C) 制御を移すリカバリ・コードを発生するステップを含み、該リカバリ・コードが前記第2命令のコピーを含む、コンピュータ読み取り可能媒体。

【請求項32】 コンピュータ・システム上で実行されたとき、格納命令と、ロード命令と、該ロード命令によって読み出されたデータに依存する少なくとも1つの計算命令とを含むソース・プログラムをコンパイルする方法を実行するコンパイラでエンコードしたコンピュータ読み取り可能媒体であって、前記方法が、

(A) 前記ロード命令および前記少なくとも1つの計算命令を前記格納命令の前にスケジュールするステップと、

(B) 前記プログラムの実行中、前記格納およびロード命令が共通メモリ位置にアクセスしたことを判定した場合に分岐する分岐命令を発生するステップと、

(C) 前記プログラムの実行中、前記格納およびロード命令が共通メモリ位置にアクセスしたことを判定した場合に前記分岐命令が分岐するリカバリ・コードを発生するステップであって、該リカバリ・コードが前記ロード命令および前記少なくとも1つの計算命令のコピーを含む、ステップと、

を含むコンピュータ読み取り可能媒体。

【請求項33】 請求項32記載のコンピュータ読み取り可能媒体において、前記ステップ(B)が、分岐し、前記プログラムの実行中に前記格納およびロード命令が共通メモリ位置にアクセスしたか否かについて判定するチェック命令を発生するステップを含む、コンピュータ読み取り可能媒体。

【請求項34】 命令実行方法であって、  
投機的とマークされた少なくとも1つの命令を実行し、  
前記少なくとも1つの命令の実行の完全性を検証し、  
前記少なくとも1つの命令の実行の完全性が検証された場合、他の命令の実行を継続し、  
前記少なくとも1つの命令の実行の完全性が検証されない場合、  
リカバリ・コードを実行し、  
前記リカバリ・コードを実行した後、前記他の命令の実行を継続する、  
ことを含む方法。

【請求項35】 コンピュータ・システム上で実行されたとき、ソース・プログラムをコンパイルして、複数の基本ブロックに編成された複数の命令を含むコンパイル化プログラムを発生する方法を実行するコンパイラでエンコードしたコンピュータ読み取り可能媒体であって、各基本ブロックが連続する命令の集合を含み、前記複数の命令が、第1基本ブロックに関連し、前記コンパイル化プログラムの実行中に例外を発生し得る第1命令を含み、前記方法が、

(A) 前記第1命令を、前記第1基本ブロックの外部で、前記第1基本ブロックに先行する少なくとも1つの命令の前にスケジュールするステップと、

(B) 前記コンパイル化プログラムの実行中、前記第1命令が例外を発生するか否かについて判定を行うチェック命令を発生し、前記第1命令が例外を発生したことを前記チェック命令が判定した場合、前記チェック命令が制御フローを変更するステップと、

を含むコンピュータ読み取り可能媒体。

【請求項36】 請求項35記載のコンピュータ読み取り可能媒体において、前記方法が、更に、前記チェック命令を前記第1基本ブロック内でスケジュー

ルするステップを含む、コンピュータ読み取り可能媒体。

【請求項37】 請求項35記載のコンピュータ読み取り可能媒体において、前記方法が、更に、前記複数の命令の各々を、投機的または非投機的として識別するステップを含む、コンピュータ読み取り可能媒体。

【請求項38】 請求項35記載のコンピュータ読み取り可能媒体において、前記方法が、更に、前記第1命令が例外を発生したと前記チェック命令が判定した場合に制御を移すリカバリ・コードを発生するステップを含み、該リカバリ・コードが前記第1命令のコピーを含む、コンピュータ読み取り可能媒体。

【請求項39】 コンピュータ読み取り可能な形態でプログラムを格納したコンピュータ読み取り可能媒体であって、前記プログラムが、

複数の基本ブロックに編成された複数の命令から成り、各基本ブロックが連続する命令の集合を含み、前記複数の命令が、

第1基本ブロックに関連し、前記プログラムの実行中に例外を発生し得る第1命令であって、前記第1基本ブロックの外部で、前記第1基本ブロックに先行する少なくとも1つの命令の前にスケジュールされた、第1命令と、

前記プログラムの実行中、前記第1命令が例外を発生するか否かについて判定を行うチェック命令であって、前記第1命令が例外を発生したと前記チェック命令が判定した場合、制御フローを変更する、チェック命令と、を含むコンピュータ読み取り可能媒体。

【請求項40】 請求項39記載のコンピュータ読み取り可能媒体において、前記チェック命令を前記第1基本ブロック内にスケジュールされる、コンピュータ読み取り可能媒体。

【請求項41】 請求項39記載のコンピュータ読み取り可能媒体において、前記複数の命令の各々を、投機的または非投機的として識別する、コンピュータ読み取り可能媒体。

【請求項42】 請求項41記載のコンピュータ読み取り可能媒体において、前記複数の命令が、更に、前記第1命令が例外を発生したと前記チェック命令が判定した場合に制御を移すリカバリ・コードを含む、コンピュータ読み取り可能媒体。

【請求項43】 請求項42記載のコンピュータ読み取り可能媒体において、前記リカバリ・コードが、前記第1命令のコピーを含む、コンピュータ読み取り可能媒体。

【請求項44】 コンピュータ・システムであって、  
複数の基本ブロックに編成された複数の命令を含むプログラムを実行する手段であって、各基本ブロックが連続する命令の集合を含み、前記複数の命令が、第1基本ブロックに関連し、前記プログラムの実行中に例外を発生し得る第1命令を含む、手段と、

前記第1命令を、前記第1基本ブロックの外部で、前記第1基本ブロックに先行する少なくとも1つの命令の前に実行する手段と、

前記第1命令が例外を発生するか否かについて判定を行う手段と、

前記第1命令が例外を発生したと判定した場合に制御フローをリカバリ・コードに変更する手段と、  
を備えるコンピュータ・システム。

【請求項45】 請求項44記載のコンピュータ・システムにおいて、前記リカバリ・コードが、前記第1命令のコピーを含む、コンピュータ・システム。

【請求項46】 コンピュータ・システムであって、  
複数の基本ブロックに編成されている複数の命令を含むプログラムに対して実行スケジュールを作成するコンパイラを備え、各基本ブロックが、連続する命令の集合を含み、前記複数の命令が、第1基本ブロックに関連し前記プログラムの実行中に例外を発生し得る第1命令を含み、前記コンパイラは、前記第1命令が前記プログラムの実行中例外を発生しないことに前記コンパイラが確信できない場合、前記第1命令を、前記第1基本ブロックの外部で、前記第1基本ブロックに先行する少なくとも1つの命令の前にスケジュールする手段を含み、更に、前記コンパイラは、前記第1命令が例外を発生したとチェック命令が判定した場合制御フローにおいて変更を含むチェック命令を発生する手段を含む、コンピュータ・システム。

【請求項47】 請求項46記載のコンピュータ・システムにおいて、前記コンパイラが、更に、前記第1命令が例外を発生した場合に制御を移すリカバリ

・コードを発生する手段を含み、該リカバリ・コードが前記第1命令のコピーを含む、コンピュータ・システム。

【請求項48】 コンピュータ読み取り可能な形態でプログラムを格納したコンピュータ読み取り可能媒体であって、前記プログラムが、

第1投機的命令を備え、該第1投機的命令の実行中に命令例外状態が生じる可能性があり、前記第1投機的命令は、前記例外状態が最初に検出された場合、命令例外の通報を延期し、該命令例外を通報せずに実行を完了する、コンピュータ読み取り可能媒体。

【請求項49】 請求項48記載のコンピュータ読み取り可能媒体において、前記第1投機的命令は第1宛先を有し、前記例外状態が最初に検出された場合、前記第1投機的命令は、前記第1宛先に、前記第1投機的命令の実行中に前記命令例外が検出されたことを示す情報を格納する、コンピュータ読み取り可能媒体。

【請求項50】 請求項49記載のコンピュータ読み取り可能媒体において、前記プログラムが、更に、前記第1投機的命令の後にスケジュールされ、前記第1宛先における結果に対して動作する第2投機的命令を含み、該第2投機的命令が第2宛先を有し、前記第1投機的命令の実行中に前記命令例外が検出されたことを示す前記情報を前記第1宛先に格納する場合、前記第2投機的命令が、前記第2宛先に、前記プログラムの実行中に前記命令例外が検出されたことを示す情報を格納する、コンピュータ読み取り可能媒体。

【請求項51】 請求項50記載のコンピュータ読み取り可能媒体において、前記プログラムが、更に、前記第1投機的命令の後にスケジュールされ、前記第1投機的命令の実行中に前記命令例外が検出されたことを判定するチェック命令を含む、コンピュータ読み取り可能媒体。

【請求項52】 請求項51記載のコンピュータ読み取り可能媒体において、前記プログラムが、更に、前記第1投機的命令の実行中に前記命令例外が検出されたことを前記チェック命令が判定した場合に制御を移すリカバリ・コードを含む、コンピュータ読み取り可能媒体。

【請求項53】 請求項51記載のコンピュータ読み取り可能媒体において

、前記プログラムが、更に、前記第2投機的命令の実行中に前記命令例外が検出されたことを前記チェック命令が判定した場合に制御を移すリカバリ・コードを含む、コンピュータ読み取り可能媒体。

【請求項54】 請求項52記載のコンピュータ読み取り可能媒体において、前記リカバリ・コードが、前記第1投機的命令のコピーを含む、コンピュータ読み取り可能媒体。

【請求項55】 コンピュータ・システム上で実行されたとき、ソース・プログラムをコンパイルして、コンパイルされたプログラムを発生する方法を実行するコンパイラでエンコードしたコンピュータ読み取り可能媒体であって、前記方法が、

(A) 第1投機的命令を発生するステップを含み、該第1投機的命令の実行中に命令例外状態が生じる可能性があり、前記第1投機的命令は、前記例外状態が最初に検出された場合、命令例外の通報を延期し、該命令例外を通報せずに実行を完了する、コンピュータ読み取り可能媒体。

【請求項56】 請求項55記載のコンピュータ読み取り可能媒体において、前記第1投機的命令は第1宛先を有し、前記ステップ(A)が、前記例外状態が最初に検出された場合、前記第1投機的命令が、当該第1投機的命令の実行中に前記命令例外が検出されたことを示す情報を前記第1宛先に格納するように前記第1投機的命令を発生するステップを含む、コンピュータ読み取り可能媒体。

【請求項57】 請求項56記載のコンピュータ読み取り可能媒体において、前記方法が、更に、第1宛先における結果に対して動作する第2投機的命令を発生するステップを含み、該第2投機的命令が第2宛先を有し、前記第1投機的命令の実行中に前記命令例外が検出されたことを示す前記情報を前記第1宛先に格納する場合、前記第2投機的命令が、前記プログラムの実行中に前記命令例外が検出されたことを示す情報を前記第2宛先に格納する、コンピュータ読み取り可能媒体。

【請求項58】 請求項57記載のコンピュータ読み取り可能媒体において、前記方法が、更に、前記第1投機的命令の実行中に前記命令例外が検出されたか否かについて判定を行うチェック命令を発生するステップを含む、コンピュー

タ読み取り可能媒体。

【請求項59】 請求項58記載のコンピュータ読み取り可能媒体において、前記方法が、更に、前記第1投機的命令の実行中に前記命令例外が検出されたことを前記チェック命令が判定した場合に制御を移すりカバリ・コードを発生するステップを含む、コンピュータ読み取り可能媒体。

【請求項60】 コンピュータ・システムであって、  
投機に基づいて第1プログラム命令を実行する手段であって、該第1プログラム命令が、当該第1プログラム命令の実行中に命令例外状態を生じる可能性がある、手段と、

前記第1プログラム命令の実行中に例外状態が検出された場合、該命令例外の通報を延期させる手段と、

前記投機が誤っていたか否かについて判定を行う手段と、

前記投機が誤っていた場合、前記命令例外を無視する手段と、

前記第1プログラム命令の実行中に前記例外状態が検出されたことが判定され、前記投機が正しかった場合、前記プログラムの制御フローをリカバリ・コードに変更する手段と、

を備えるコンピュータ・システム。

【請求項61】 請求項60記載のコンピュータ・システムであって、更に、前記投機が正しかった場合、前記命令例外を通報する手段を含む、コンピュータ・システム。

【請求項62】 コンピュータ・システム上で実行されたとき、以下のステップを含む方法を実行するプログラムでエンコードしたコンピュータ読み取り可能媒体であって、前記方法が、

投機に基づいて第1プログラム命令を実行するステップであって、該第1プログラム命令が、当該第1プログラム命令の実行中に命令例外状態を生じる可能性がある、ステップと、

前記第1プログラム命令の実行中に例外状態が最初に検出された場合、該命令例外の通報を延期させるステップと、

前記投機が誤っていたか否かについて判定を行うステップと、

前記投機が誤っていた場合、前記命令例外を無視するステップと、  
前記第1プログラム命令の実行中に前記例外状態が検出されたことが判定され、前記投機が正しかった場合、前記プログラムの制御フローをリカバリ・コードに変更するステップと、  
を含むコンピュータ読み取り可能媒体。

【請求項63】 請求項62記載のコンピュータ読み取り可能媒体であって、前記方法が、更に、前記投機が正しかった場合、前記命令例外を通報するステップを含む、コンピュータ読み取り可能媒体。

【請求項64】 コンピュータ読み取り可能な形態でプログラムを格納したコンピュータ読み取り可能媒体であって、前記プログラムが、  
オペランドを受け取り、命令例外状態を生じる可能性がある第1投機的命令を備え、該第1投機的命令が第1タイプのオペランドを受け取った場合、実行し結果を計算し、更に、前記第1投機的命令が例外トークンを含む第2タイプのオペランドを受け取った場合、前記命令を実行せずに単に前記トークンを前記投機的命令の宛先に渡し、これにより、前記第1投機的命令は、前記例外状態が最初に検出されたときに、命令例外の通報を延期させる、  
コンピュータ読み取り可能媒体。

【請求項65】 第1命令と該第1命令に係る第2命令とが共通メモリ位置にアクセスするか否かについて判定を行う方法であって、  
前記第1命令を実行するステップと、  
アドレス・テーブル内のエントリにおいて、前記第1命令を一意的に識別するステップと、

前記第2命令を実行するステップであって、該第2命令の実行が、前記アドレス・テーブルを探索し、前記第1命令を識別するエントリを求めることを含む、ステップと、

前記第1命令のメモリ空間範囲が、前記第2命令のメモリ空間範囲と重複する場合、前記アドレス・テーブルから前記第1命令を識別するエントリを除去するステップと、

前記第1および第2命令によって共通メモリ位置がアクセスされたか否かにつ

いて判定を行うチェック命令を実行するステップであって、該チェック命令が前記アドレス・テーブルを探索して前記第1命令に対応するエントリを求め、前記第1命令に対するエントリが前記アドレス・テーブル内で見つからない場合、前記第1および第2命令によって共通メモリ位置がアクセスされたと判定する、ステップと、  
を含む方法。

【請求項66】 請求項65記載の方法において、チェック命令を実行する前記ステップが、共通メモリ位置がアクセスされた場合、リカバリ・コードに分岐するステップを含む、方法。

【請求項67】 請求項65記載の方法において、チェック命令を実行する前記ステップが、前記第1命令に対するエントリが見つかった場合、共通メモリ位置はアクセスされなかったと判定するステップを含む、方法。

【請求項68】 コンピュータ・システムであって、  
メモリと、  
アドレス・テーブルと、  
前記メモリ内にコンピュータ読み取り可能な形態で格納されているコンパイルされたプログラムと、  
を備え、前記プログラムが、  
格納命令と、  
前記格納命令の前にスケジュールされたロード命令と、  
前記アドレス・テーブルにアクセスし、前記格納命令および前記ロード命令がメモリ内の共通位置にアクセスしたか否かについて判定するチェック命令と、  
を含み、前記格納命令の実行後、前記アドレス・テーブル内に前記ロード命令を識別するエントリがない場合、前記格納およびロード命令によってメモリ内の共通位置がアクセスされたことを示す、  
コンピュータ・システム。

【請求項69】 請求項68記載のコンピュータ・システムにおいて、前記アドレス・テーブルが、前記ロード命令のために用いられるレジスタ数に対応する数のエントリを含む、コンピュータ・システム。

## 【発明の詳細な説明】

## 【0001】

関連出願に対する引用

本願は、1997年10月13日に出願された米国特許出願第08/953,836号の一部継続出願である。

発明の分野

本発明は、コンピュータ・システムにおける命令の実行に関する。本発明の一態様は、投機的に (speculatively) 実行したコンピュータ命令によって生じた例外の回復に関する。本発明の別の態様によれば、命令の実行およびそれに依存する計算を順不同に進めて性能の向上を図ることに関する。

関連技術の説明

「基本ブロック (basic block)」とは、ブランチ (分岐) および／またはブランチ・ターゲットによって境界が示される、ブランチやブランチ・ターゲットを含まない隣接 (連続) する命令の集合 (セット) である。このことは、基本ブロック内のいずれかの命令が実行されると、当該基本ブロック内の全命令が実行される、即ち、いずれの基本ブロック内に含まれる命令は、全てか皆無か (all-or-nothing) を基本として実行されるということを意味する。基本ブロック内にある命令は、当該基本ブロックを目標 (ターゲット) に設定する以前のブランチによって制御が当該基本ブロックに進んだときに、イネーブルされ実行される (ここで用いる場合、「ターゲット設定」 (targeting) とは、ブランチ選択による明示的なターゲット設定およびブランチ選択によらない暗示的なターゲット設定双方を含むものとする)。前述の説明では、制御が基本ブロックに進んだ場合、当該基本ブロック内の全命令を実行しなければならず、制御が基本ブロックに行かない場合、基本ブロック内の命令は全く実行されないことを意味する。制御が命令に進む前に、当該命令を実行すること、または実行を指定することを「投機」 (speculation) と呼ぶ。プログラムの実行中にプロセッサが行う投機を「動的投機」と呼び、コンパイラが指定する投機を「静的投機」と呼ぶ。動的投機は従来技術では公知である。静的投機については、従来技術の大半はこれに基づいておらず、言及もしていないが、

近年静的投機に対する論及も表面化し始めている。

#### 【0002】

2つの命令の内、一方が他方の結果を必要としない場合、これらを「独立」と呼び、一方の命令が他方の命令の結果を必要とする場合、「従属」と呼ぶ。独立命令は、並列に実行可能であるが、従属命令は直列的に実行しなければならない。プログラムの性能は、独立命令を識別し、これらをできるだけ並列に実行することによって向上する。経験によって、多数の基本ブロックを越えて探索（サーチ）する方が、個々の基本ブロック内だけを探索するよりも、多くの独立命令を発見できることが示されている。しかしながら、多数の基本ブロックからの命令を同時に実行するには、投機が必要となる。

#### 【0003】

独立命令を識別しスケジューリングすることによって、性能向上を図ることは、コンパイラおよびプロセッサの主要なタスクの1つである。コンパイラおよびプロセッサ設計におけるトレンドは、独立命令の探索範囲を連続的な発生毎に拡大することである。従来技術の命令セットでは、例外を発生し得る命令は、コンパイラによって投機（推測）することができない。何故なら、その命令が例外を生ずると、プログラムは誤った挙動を行う虞れがあるからである。このため、コンパイラの独立命令探索に有用な範囲が限定されてしまい、投機はプロセッサが動的投機によってプログラムの実行時に行わざるを得ない。しかしながら、動的投機は、大量のハードウェアの複雑化を伴い、動的投機を適用する基本ブロックの数と共に指数的に増大する。このために、動的投機の範囲は実際上限定されることになる。対照的に、コンパイラが独立命令を探索することができる範囲は遥かに広く、潜在的には（可能性として）プログラム全体に及ぶ。更に、一旦単一の基本ブロック境界を越えて静的投機を実行するようにコンパイラを設計すると、数箇所の基本ブロックの境界を越えて静的投機を行っても、追加される複雑化は殆ど生じない。

#### 【0004】

静的投機に着手する場合、いくつかの問題を解決しなければならない。最も重要な問題の1つは、静的投機した命令が遭遇する例外状態の処理である。先に注

記したように、投機的命令の例外は、当該命令の実行時には引き渡す (deliver) ことができないので、命令を投機した基本ブロック (「原始 (originating) 基本ブロック」として知られている) に制御を渡すまで、例外の引き渡しを延期するようなコンパイラ・ビジブル機構 (compiler-visible mechanism) が望ましい。同様の機能を実行する機構が従来技術にも存在し、動的に投機した命令の例外を遅らせ、後になってから引き渡している。しかしながら、定義上、この機構はコンパイラにはビジブル (可視) でなく、したがってコンパイラによって操作することができず、コンパイラ主導の投機においてその役割を果たすことができない。静的に投機した命令の致命的 (フェイタル) または非致命的例外を遅らせ、後に引き渡すことは、従来技術における公知の方法および装置では不可能であった。しかしながら、従来技術には、以下のような静的投機の限定的な形態は存在する。(1) 例外状態の遅滞および後の回復を伴わない形態、(2) および本発明の幅 (breadth) および範囲に及ぶ静的投機を可能としない形態。

#### 【0005】

したがって、静的投機に着手する場合、当技術分野では、投機的命令の例外を処理し、投機的命令の副作用がプログラマには全く見えないようにする機構が求められている。更に、この機構は、できるだけ多くの形態の静的投機に適用できるようにでなければならない。

#### 【0006】

また、できるだけ多くの独立命令の並列実行を可能にすることにより、コンピュータ・システムにおける性能向上を図る機構も求められている。これは、第2命令およびそれに依存する計算を、第1命令の例外に依存する可能性があるデータに対して行う可能性がある場合でも望ましいものである。

#### 発明の概要

本発明の例示的な実施形態の1つでは、コンパイルされたプログラムをコンピュータ読み取り可能な形態で格納 (記憶) したコンピュータ読み取り可能媒体を提供する。プログラムは、格納 (ストア) 命令と、この格納命令の前にスケジュールされたロード命令と、ロード命令によって読み取られるデータに依存し、格

納命令よりも前にスケジュールされる計算命令と、格納命令およびロード命令がメモリ内の共通位置にアクセスするか否かについて判定を行うチェック命令とを含む。

#### 【0007】

本発明の別の例示的な実施形態では、メモリと、格納命令を実行する手段と、格納命令の前にロード命令を実行する手段と、格納命令の前に、ロード命令によって読み取られたデータに依存する計算命令を実行する手段とを含むコンピュータ・システムを提供する。また、コンピュータ・システムは、格納命令およびロード命令がメモリ内の共通位置にアクセスしたか否かについて判定を行う手段も含む。

#### 【0008】

本発明の別の例示的な実施形態では、格納命令、ロード命令、およびロード命令によって読み取られるデータに依存する計算命令を含むソース・プログラムに対して実行スケジュールを作成するコンパイラを含むコンピュータ・システムを提供する。コンパイラは、格納命令およびロード命令がプログラムの実行中共通メモリ位置にアクセスしないことにコンパイラが確信できない場合、ロード命令および計算命令を格納命令よりも前にスケジュールする手段を含む。

#### 【0009】

別の例示的な実施形態では、コンピュータ読み取り可能な形態でコンパイルされたプログラムを格納したコンピュータ読み取り可能媒体を提供する。プログラムは、格納命令と、格納命令の前にスケジュールされたロード命令と、プログラムの実行中、格納命令およびロード命令が共通メモリ位置にアクセスするか否かについて判定を行うチェック命令を含む。チェック命令は、格納命令およびロード命令が共通メモリ位置にアクセスしたと判定した場合、制御フローを変更する。

#### 【0010】

別の例示的な実施形態では、メモリと、格納命令および以前に実行したロード命令がメモリ内の共通位置にアクセスしたか否かについてチェックする手段と、格納命令およびロード命令がメモリ内の共通位置にアクセスしたと判定した場合

、制御フローをリカバリ（回復）コードに変更する手段とを含むコンピュータ・システムを提供する。

#### 【0011】

別の例示的な実施形態では、コンピュータ・システム上で実行されるとき、格納命令と、ロード命令と、このロード命令によって読み取られたデータに依存する計算命令とを含むソース・プログラムをコンパイルする方法を実行するコンパイラによってエンコードしたコンピュータ読み取り可能媒体を提供する。前記方法は、プログラムの実行中、格納およびロード命令が共通メモリ位置にアクセスしないか否かについて判定を行うステップと、格納およびロード命令が共通メモリ位置にアクセスしないと判定できない場合、ロード命令および計算命令を格納命令の前にスケジュールするステップとを含む。

#### 【0012】

別の例示的な実施形態では、コンピュータ・システム上で実行されるとき、第1命令および第2命令を含むプログラムをコンパイルする方法を実行するコンパイラであって、第2命令が第1命令の実行に依存するデータ上で動作しないことに確信できないコンパイラによってエンコードしたコンピュータ読み取り可能媒体を提供する。前記方法は、第2命令を第1命令の前にスケジュールするステップと、プログラムの実行中、第2命令が、第1命令の実行に依存するデータ上で動作するか否かについて判定を行うチェック命令を発生するステップとを含む。

#### 【0013】

更に別の例示的な実施形態では、コンピュータ・システム上で実行されるとき、ロード命令と、格納命令と、ロード命令によって読み取られたデータに依存する計算命令とを含むソース・プログラムをコンパイルする方法を実行するコンパイラによってエンコードされたコンピュータ読み取り可能媒体を提供する。前記方法は、ロード命令および計算命令を格納命令の前にスケジュールするステップと、プログラムの実行中、格納およびロード命令が共通メモリ位置にアクセスしたことを判定した場合に分岐する分岐命令を発生するステップとを含む。前記方法は、更に、プログラムの実行中、格納およびロード命令が共通メモリ位置にアクセスしたことを判定した場合に分岐命令が分岐するリカバリ・コードを発生す

るステップを含み、リカバリ・コードは、ロード命令および計算命令のコピーを含む。

【0014】

別の例示的な実施形態は、命令実行方法に関し、投機的と印（マーク）されている少なくとも1つの命令を実行し、少なくとも1つの命令の実行の完全性（integrity）を検証し、少なくとも1つの命令の実行の完全性が検証された場合、他の命令の実行を継続し、少なくとも1つの命令の実行の完全性が検証されない場合、リカバリ・コードを実行し、リカバリ・コードを実行した後、他の命令の実行を継続する命令から成る。

【0015】

本発明の更に別の例示的な実施形態は、コンピュータ・システム上で実行されるとき、ソース・プログラムをコンパイルして、複数の基本ブロックに編成された複数の命令を含むコンパイルされたプログラムを発生する方法を実行するコンパイラによってエンコードされたコンピュータ読み取り可能媒体に関する。各基本ブロックは、連続する命令の集合を含み、複数の命令が、第1基本ブロックに関連し、コンパイルされたプログラムの実行中に例外を発生し得る第1命令を含む。前記方法は、（A）第1命令を、第1基本ブロックの外部で、第1基本ブロックに先行する少なくとも1つの命令の前にスケジュールするステップと、（B）コンパイルされたプログラムの実行中、第1命令が例外を発生するか否かについて判定を行うチェック命令を発生するステップとを含む。

【0016】

本発明の別の例示的な実施形態は、コンピュータ読み取り可能な形態でプログラムを格納したコンピュータ読み取り可能媒体に関する。前記プログラムは、複数の基本ブロックに編成された複数の命令から成り、各基本ブロックが連続する命令の集合を含む。複数の命令は、第1基本ブロックに関連し、プログラムの実行中に例外を発生し得る第1命令であって、第1基本ブロックの外部で、第1基本ブロックに先行する少なくとも1つの命令の前にスケジュールされる第1命令と、プログラムの実行中、第1命令が例外を発生するか否かについて判定を行うチェック命令とを含む。

## 【0017】

本発明の更に別の例示的な実施形態は、コンピュータ・システムに関し、複数の基本ブロックに編成された複数の命令を含むプログラムを実行する手段であって、各基本ブロックが連続する命令の集合を含み、複数の命令は、第1基本ブロックに関連し、プログラムの実行中に例外を発生し得る第1命令を含む、手段と、第1命令を、第1基本ブロックの外部で、第1基本ブロックに先行する少なくとも1つの命令の前に実行する手段と、第1命令が例外を発生したか否かについて判定を行う手段とを備える。

## 【0018】

本発明の別の例示的な実施形態は、コンピュータ・システムに関し、複数の基本ブロックに編成されている複数の命令を含むプログラムに対して例外スケジュールを作成するコンパイラを備え、各基本ブロックが、連続する命令の集合を含み、複数の命令が、第1基本ブロックに関連し、プログラムの実行中に例外を発生し得る第1命令を含む。コンパイラは、第1命令がプログラムの実行中例外を発生しないことに当該コンパイラが確信できない場合、第1命令を、第1基本ブロックの外部で、第1基本ブロックに先行する少なくとも1つの命令の前にスケジュールする手段を含む。

## 【0019】

本発明の更に別の例示的な実施形態は、コンピュータ読み取り可能な形態でプログラムを格納したコンピュータ読み取り可能媒体に関する。前記プログラムは、第1投機的命令を備え、この第1投機的命令の実行中に命令例外状態が発生する可能性があり、第1投機的命令は、例外状態が最初に検出された場合、命令例外の通報を延期し、命令例外を通報せずに実行を完了する。

## 【0020】

本発明の別の例示的な実施形態は、コンピュータ・システム上で実行されるとき、ソース・プログラムをコンパイルして、コンパイルされたプログラムを発生する方法を実行するコンパイラによってコード化されたコンピュータ読み取り可能媒体に関する。前記方法は、(A)第1投機的命令を発生するステップから成り、第1投機的命令の実行中に命令例外状態が発生する可能性があり、第1投機

的命令は、例外状態が最初に検出された場合、命令例外の通報を延期し、命令例外を通報せずに実行を完了する。

#### 【0021】

本発明の更に別の例示的な実施形態は、コンピュータ・システムに関し、投機に基づいて第1プログラム命令を実行する手段であって、第1プログラム命令が、当該第1プログラム命令の実行中に命令例外状態を生じる可能性がある、手段と、第1プログラム命令の実行中に例外状態が検出された場合、命令例外の通報を延期させる手段と、投機が誤っていたか否かについて判定を行う手段と、投機が誤っていた場合、命令例外を無視する手段とを備えている。

#### 【0022】

本発明の別の例示的な実施形態は、コンピュータ・システム上で実行された場合、以下のステップを含む方法を実行するプログラムをエンコードしたコンピュータ読み取り可能媒体に関し、前記方法は、投機に基づいて第1プログラム命令を実行するステップであって、第1プログラム命令が、当該第1プログラム命令の実行中に命令例外状態を発生する可能性がある、ステップと、第1プログラム命令の実行中に例外状態が最初に検出された場合、命令例外の通報を延期させるステップと、投機が誤っていたか否かについて判定を行うステップと、投機が誤っていた場合、命令例外を無視するステップとを含む。

#### 詳細な説明

本発明の一実施形態は、繰り上げた（進めた）命令即ち投機命令の実行中に遭遇した問題からの回復を可能とする方法および装置を対象とする。本発明のこの態様は、あらゆる種類のコンピュータ・システムでも採用可能である。かかるコンピュータ・システムの一例が、図1に示す汎用コンピュータ50である。汎用コンピュータ50は、プロセッサ52、入力デバイス54、出力デバイス56、およびメモリ58を含み、これらはバス60を通じて接続されている。メモリ58は、主メモリ62（即ち、ダイナミック半導体メモリのような高速揮発性メモリ）および副メモリ64（即ち、磁気ディスクのような不揮発性メモリ）を含む。メモリ58は、プロセッサ52上で実行する1つ以上のプログラム66を格納する。

## 【0023】

プログラム66は、プロセッサ52によって実行されると、汎用コンピュータ50を制御する。プログラム66は、コンパイラを含み、その機能については図6に関連して以下で説明する。

## 【0024】

尚、図1のコンピュータ・システム50は、単に例示の目的で提示するに過ぎず、以下に記載する本発明の実施形態は、多数のその他の種類および構成のコンピュータ・システム上でも実現可能であることは認められよう。本発明の一態様は、静的投機命令の実行中に遭遇した問題から回復する方法および装置を提供する。本発明の一実施形態は、コンパイラによって投機的に実行するようにスケジュール化されたあらゆる形式の命令セグメントを実行し、投機的に実行された命令の実行の完全性を検証し、何らかの問題が検出された場合、問題を補正するリカバリ・コードを実行することである。

## 【0025】

命令は、投機的および非投機的の2種類に分類される。コンパイルの開始時に、全ての命令は非投機的に初期化される。スケジューリングの間に、コンパイラが命令の原始基本ブロック以外の命令をスケジュールする場合、コンパイラはこの命令に投機的であるというマークを付ける。例外状態に遭遇した非投機的命令は例外を発生する。例外状態に遭遇した投機的命令は、例外を発生しないが、代わりに「延期例外トークン」(DET: deferred exception token)をその宛先(行先)に書き込む。例外状態の存在により、指定したコンピュータ命令が適正なオペランドで完了することが妨げられ、したがって、この命令の宛先は、正しい結果の代わりに、DETを含むことになる。DETを読み取った非投機的命令は例外を発生する。DETを読み取った投機的命令は、別のDETを当該命令の宛先に書き込む(この場合にも宛先には正しい結果が収容されないことを注記しておく)。この挙動のことを「伝搬」(propagation)と呼ぶ。特定の投機的命令の原始基本ブロック内に非投機的命令を置くことにより、そして投機的命令の宛先(またはDETを伝搬した可能性があるいずれかの位置)を読み取るように非投機的命令を構成することにより、制御

が原始基本ブロックに進んだ時点で、投機的命令が発生したDETを検出することができる。この時点で、元来DETが発生する原因となった例外状態を再現し、以前に伝搬したDET全てを正しい結果と置き換える必要がある。これは、「回復（リカバリ：recovery）」と呼ばれるプロセスによって行われる。回復は、コンパイラが発生する追加のコードによるプログラムの改良を含むことができ、このコードは、非投機的形態の依存性投機的命令集合（セット）のコピーであり、実行時に、全ての例外状態が例外が発生し、以前に書き込んだ宛先の全てを正しい結果で上書きする。リカバリ・コードは、命令シーケンスの正確なコピーである必要はなく、実行すると同じ結果が得られるコードであればよい。更に、本発明の一実施形態では、DETの存在をチェックし、DETが検出された場合に関連するリカバリ・コードを活性化するという特定した目的で、新たな命令を定義する。

#### 【0026】

ここまでに説明した本発明の実施形態は、DETの正確な形態には依存しない。また、本発明の精神または範囲に影響を与えることなく、投機的および非投機的命令を指定する代替実施形態も可能である。例えば、ある命令がその原始基本ブロックの外部でスケジュールされたか否かには無関係に、投機的に振る舞うようにその命令を定義することも可能である。

#### 【0027】

この時点までに引用した投機を、「制御投機（control speculation）」と呼ぶ。何故なら、命令が実行された後に、制御がこれらに移されるからである。投機は、制御投機以外にも、他の形態を取ることができる。その一例は、「データ投機（data speculation）」であり、これによって、命令Bに依存し得る命令Aを、命令Bの前に実行することを可能とする機構を定義する。データ投機はいずれの命令クラスにも適用可能であるが、以下ではデータ投機を例証するために、ロードおよび格納について説明する。格納の下にあるロードは、このロードによって読み取られるアドレスが、格納によって書き込まれるアドレスとは決して等しくないことを示すことができない限り、一般には、この格納の上にはスケジュールすることはできない。アドレスが等し

い場合、ロードは当然格納の結果を受け取ることになる。しかしながら、ロードによって読み取られるアドレスが、格納によって書き込まれるアドレスとは決して等しくなることを示せば、格納の上に安全にロードをスケジュールすることができる。データ投機が発生するのは、双方によってアクセスされるアドレスが決して等しくなることを証明できないときに、コンパイラが格納よりも上にロードをスケジュールした場合である。双方の命令によってアクセスされるアドレスが等しいと実行時に判定された場合、衝突として知られるエラー状態が発生する。衝突の場合、回復機構を用いて、誤って書き込まれた宛先を全て訂正する。本発明の一実施形態では、ロード命令およびこれに依存する1つ以上の命令は、ロード命令と格納命令との間に衝突があり得るとコンパイラが判断した場合でも、コンパイラによって格納命令の上にスケジュールする。このように、本発明の態様は、制御投機、データ投機、およびその他の投機形態を対象とする。

#### 【0028】

本発明の一態様は、コンパイラが命令を投機的に実行するようにスケジュールすることができ、しかもコンピュータ・システムは、命令の投機的実行の間に発生する投機エラーから回復することができる技法を提供する。本発明の別の態様は、命令を順不同に進める (advance) 方法および装置を対象とする。これは、第1および第2命令がメモリの一部において同じアドレスにアクセスすることによる衝突として知られているエラー状態を生ずる可能性がある第1命令 (例えば、格納命令) よりも先に、第2命令およびこれに依存する計算全体 (例えば、ロード命令) を実行するようにスケジュールすることを含む。

#### 【0029】

本発明の態様の幾つかを実現するために、コンパイラが、命令をその原始基本ブロックの外部でスケジュールし (制御投機)、同じメモリ位置にアクセスする潜在的な可能性があり、したがって潜在的に依存性のある命令の並列実行をスケジュールする (データ投機) ことを可能にする、コンピュータ・アーキテクチャを規定することができる。かかるコンピュータ・アーキテクチャの一例が、1997年10月13日に出願した、Jonathan K. Ross et al (ジョナサン K. ロスその他) による "COMPUTER ARCHIT

ECTURE FOR THE DEFERRAL OF EXCEPTION  
S ON SPECULATIVE INSTRUCTION" (投機的命令の例外延期のためのコンピュータ・アーキテクチャ) と題する同時係属中の米国特許出願第08/949, 295号に、更に詳細に記載されている。その内容は、この言及により本願にも援用されるものとする。以下では、本発明の態様をこのアーキテクチャに関して記載するが、本発明はこのアーキテクチャとともに使用することに限定される訳ではなく、他のアーキテクチャ構造を用いても実現可能である。これについては、以下で更に詳しく説明する。

#### 【0030】

この新たなアーキテクチャは、例外状態が発生した場合に直ちに例外を通報しない「投機的」命令集合を定義する。代わりに、投機的命令が、「延期例外トークン」(DET)を、当該命令が指定する宛先に書き込むことによって、例外を延期させる。また、命令集合は、「非投機的命令」も含み、これは、従来の命令と同様に、例外状態が発生した場合直ちに例外を通報する。

#### 【0031】

命令例外は、当技術分野では周知のことであり、ページ・フォールト、不正オペランド、特権的違反、ゼロによる除算演算、オーバーフロー等が含まれるが、これらに限定される訳ではない。また、新たなアーキテクチャは、新たな種類のメモリ投機も提供し、プログラマによって定義された論理的順序において格納命令の後に続くロード命令は、同じメモリ位置をアクセスしないであろうという投機に基づいて、これら2つの命令を格納命令の前に実行することができる。例えば、最新の投機的メモリ・アクセスの記録を収容する繰り上げロード・アドレス・テーブル (ALAT: advanced load address table) にアクセスすることができるメモリ投機チェックを備え、投機が正しいか否かについて判定を行う。投機が正しい場合、命令は適正に実行されたことになる。正しくない場合、ロード命令、およびこのロード命令に依存し格納よりも上にスケジュールされたいずれの命令も、再実行し格納命令によって書き込まれた内容を引き出す。

#### 【0032】

投機的とマークされた命令を用いることにより、コンパイラは、それらの原始基本ブロックの外部で、命令をスケジュールすることができ、依存性である可能性のあるメモリ・アクセスを並列にスケジュールすることができる。前述のように、投機的命令が例外を発生した場合、「延期例外トークン」を、命令が指定する宛先に書き込むことができる。いずれのソースにおいてもDETを検出したあらゆる投機的命令は、その宛先にDETをコピーする。投機的命令がソースにおいてDETを発見した場合、その命令に関連する機能を実行する必要はないことを注記しておく。命令は単にDETを宛先にコピーすることができるだけである。このように、DETは投機的命令のブロック全体を伝搬する。したがって、本発明の一実施形態では、計算結果を含む宛先にDETがあるか否かチェックすれば、計算に用いられる各オペランドをチェックしなくても済む。

#### 【0033】

ソースにおいてDETを検出したあらゆる非投機的命令は、即時例外を発生することができる。したがって、DETは、非投機的命令に辿り着くまで（そして辿り着けば）、データフロー状に投機的命令中を伝搬する。

#### 【0034】

実行時に、命令を実行した投機が正しくないことをプログラムが判断した場合（例えば、誤って予測されたブランチ）、プログラムは単にDETを無視すればよい。何故なら、DETはプログラムによってアクセスされないからである。しかしながら、投機が正しかった場合、DETは、当該DETを発生させた命令の原始基本ブロックを実行すれば、そして実行するときに、実際の例外に変換される。一実施形態では、この変換は、「投機チェック」命令、または、簡略化して「chk. s」と呼ばれる命令によって実行する。chk. s命令はソースを読み取り、ソースがDETを含む場合、リカバリ・コードを実装する指定のターゲット・アドレスに分岐する。同様に、本発明の一実施形態では、メモリ投機の正確性を、chk. a命令と呼ぶ「アドバンス・チェック」(advance check)命令によって判定することができる。chk. a命令は、メモリ位置が順不同でアクセスされたか否かについて判定を行い、そのようにアクセスされた場合、chk. a命令は、リカバリ・コードを実施する指定のターゲット・ア

ドレスに分岐する。chk. a 命令については、以下で更に詳しく説明することにする。chk. s および chk. a は、各々、多数の方法で実現可能であり、実行する命令の制御フローにおいて変化を生ずる。例えば、各々条件付き分岐命令として、あるいは例外ハンドラを呼び出す例外を発生する命令として、実現することができる。

#### 【0035】

定義として、chk. s および chk. a 命令は常に非投機的とする。一般に、これらの命令がDETまたは正しくないメモリ投機を検出した場合、リカバリ・コードを実行する。これは、違反命令 (offending instruction) の非投機的バージョンを含む。DETを検出したchk. s 命令に関して、リカバリ・コードの実行時に、違反命令の非投機的バージョンは、その宛先において、DETを正しい結果と置換するか、および／または例外を発生する。後のいずれかの投機的命令が違反命令に依存した場合、これらもリカバリ・コード内に含まれ、再実行される。何故なら、DETは以降の投機的命令の宛先に伝搬され、したがってこれらの宛先には正しくない結果が含まれている可能性があるからである。chk. a 命令に関して、リカバリ・コードは、違反ロード命令を再実行し、メモリから適正な内容をロードしなければならない。加えて、ロードが依存する格納よりも上でスケジュールされ、違反ロード命令に依存するあらゆる命令も、再実行される。ロード命令、および格納命令よりも上でロードされた値に依存する計算命令のスケジューリングについては、以下で更に説明することにする。違反命令に依存しないあらゆる命令は、誤ってプログラム状態を変更するので、再実行しない。コンパイラは投機的命令および投機のチェックをスケジュールしたので、コンパイラは個々の投機命令集合に適したリカバリ・コードを発生することができる。

#### 【0036】

本発明の一態様は、投機的実行のために命令をスケジュールし、適切なりカバリ・コードを発生することができるコンパイラ、および前述のアーキテクチャを実現するコンピュータ・システムのような、投機的とマークされている命令を実行可能なアーキテクチャを有するコンピュータ・システムによって実現すること

ができる。

#### 【0037】

図2は、3つの基本ブロックA1、B1およびC1で構成された元のコード・シーケンス10を示す。元のコード・シーケンス10は、プログラマが指定したコードを表わす。コード10内では、命令I0は、命令I2より前に来る命令を表わす。命令I2は、レジスタr0の内容が非ゼロである場合に命令I14に分岐する、分岐命令である。命令I4は、レジスタr1に、レジスタr2が指し示すメモリ位置の内容をロードする。命令I6は、レジスタr1の内容を3ビット位置だけシフトし、その結果をレジスタr3に書き込む。命令I8は、レジスタr3およびr5の内容を加算し、結果をレジスタr4に書き込む。命令I10は、レジスタr4の内容を、レジスタr7の内容と比較する。レジスタr4の内容がレジスタr7の内容よりも大きい場合、非ゼロ値をレジスタr6に書き込む。それ以外の場合、レジスタr6にゼロを書き込む。命令I12は、レジスタr6の内容が非ゼロの場合に、命令I100（図2には示さない）に分岐する分岐命令である。最後に、命令I14は、分岐が行われない場合に命令I12の後に来る命令群を表わす。基本ブロックB1内では、命令I12は命令I10に依存し、一方命令I10は命令I8に依存し、一方命令I8は命令I6に依存し、一方命令I6は命令I4に依存する。

#### 【0038】

図3は、本発明の例示としての一実施形態による静的投機を用いて、図2の元のコード10をスケジュールして得られた、スケジュール化コード・シーケンス20を示す。図3において、命令I4、I6、およびI8は、それらの原始基本ブロックB1の外部で、そしてブロックA1内でスケジュールされており、したがってコンパイラによって投機的とマークされている（「.s」変更子によって示す）。命令I10およびI12は、その原始基本ブロックB1の外部でスケジュールされておらず、これらは投機的ではないので「.s」を付されていない。

#### 【0039】

本発明の一実施形態では、ある種の命令、一般的に例外を発生しない命令は、常に、それらがその原始基本ブロックの外側でスケジュールされたか否かには無

関係に、投機的であるかのように振る舞う（そして、例えば、DETを伝搬する）。したがって、これらの命令には、明示的に投機的とも非投機的ともマークされていない。ロード命令のように、例外を生ずるある種の別の命令は、投機的および非投機的双方の場合があり得る。したがって、コンパイラは、これらがどのようにスケジュールされているかに応じて、投機的または非投機的と明示的にマークを付ける。また、本発明は、全ての命令に明示的かつ個別に投機的または非投機的とマークを付けるというような代替実施形態にも適用される。

#### 【0040】

最も早い投機的命令で始まり、最も遅い投機的命令で終わり、全てが同じ基本ブロックからのものである依存投機的命令のシーケンスを、「投機的依存チェーン」(speculative dependence chain)と呼ぶ（ここで用いる場合、「早い」および「遅い」は、元のプログラムの順序によって定義する）。図2および図3に示すコードでは、投機的依存コードは、命令I4で始まり、命令I6を含み、命令I8で終わる。投機的依存チェーンにおけるいずれかの命令が例外状態に遭遇した場合、違反命令の宛先にDETを書き込み、DETは投機的依存チェーンを下って伝搬する。例えば、命令I4が、ページ・フォールトのような例外状態に遭遇した場合、DETをレジスタr1に書き込む。命令I6は、レジスタr1からDETを読み取ると、次にレジスタr3にDETを書き込む。同様に、命令I8は、レジスタr3内のDETを読み取ると、次にDETをレジスタr4に書き込む。この例では、命令I6は、shl.s命令によって指定されるシフト機能を実行する必要がなく、命令I8は、add.s演算によって指定される加算機能を実行する必要がないことを注記しておく。この命令は、単に延期例外トークンを伝搬するだけでよい。したがって、一旦延期例外トークンを発生したなら、そうでない場合に投機的命令の実行によって費やされる実行資源が、他の命令を実行するために使用可能となり、あるいは休止状態のまま残しておくことにより電力消費を削減することができる。

#### 【0041】

命令I2において、レジスタr0を評価する。レジスタr0が非ゼロである場合、実行は命令I14に分岐し、この場合、レジスタr4に格納されている値は

不要となる。何故なら、命令I 4、I 6、およびI 8は、誤った投機に基づいて実行され、命令I 4、I 6またはI 8によって例外が発生されても、いずれも無視することができるからである。コンパイラは、命令I 14およびこれに続く命令が、命令I 4、I 6、およびI 8が実行されなかった場合にのみ実行されることを把握しているので、命令I 14およびこれに続く命令は、単にレジスタr 1、r 3、およびr 4内に置かれている結果を無視し、これらのレジスタを他の目的のために再利用することができる。誤った投機のために投機的に実行された命令の影響に適正に対処するコードを発生するのは、コンパイラの役割である。

#### 【0042】

しかしながら、レジスタr 0がゼロである場合、命令I 4、I 6、およびI 8の結果は妥当性が検査される。コンパイラによるスケジューリングの間、最初の命令が特定の基本ブロックから投機的とされた場合（この例では命令I 4）、コンパイラはchk. s命令（図3における命令I 9）を発行し、当該基本ブロック（この例ではB 1）内に置く。先に注記したように、chk. sは非投機的であり、それが置かれている基本ブロックの外側ではスケジュールされていない。命令I 9におけるchk. sは、命令I 8の宛先レジスタである、レジスタr 4を読み取る。命令I 9は、命令I 9によって宛先が読み取られた命令、即ち、命令I 4、I 6、およびI 8を含む、前述の投機的依存チェーンにおける全命令の結果を検証する。

#### 【0043】

命令I 4、I 6、およびI 8の実行によってDETが発生しなかった場合、命令I 4、I 6、およびI 8は妥当性が確認されたことになり、これらの命令の投機的実行が成功したことがこうして確認される。したがって、実行は命令I 10に進む。

#### 【0044】

しかしながら、命令I 4、I 6、およびI 8がDETが発生した場合、このDETはレジスタr 4に伝搬し、ここで命令I 9がこのDETを検出する。投機的依存チェーン内の命令（命令I 4、I 6、およびI 8）は、それらの宛先レジスタが正しい結果の代わりにDETを含んでいるので、信頼性のない結果を生成す

ることになる。したがって、chk. s 命令 (I 9) はDETを検出し、命令 I 4 r において開始するリカバリ・コードに分岐する。命令 I 4 r、I 6 r、および I 8 r は、それぞれ、命令 I 4、I 6、および I 8 の非投機的バージョンであり、命令 I 9 r は命令 I 9 に分岐し、chk. s 命令を再実行する。命令 I 9 は常に再実行する必要はないが、投機的依存チェーンが互いに依存しあう場合のように、そうすることが好ましい状況は多い。

#### 【0045】

命令 I 4 r、I 6 r、および I 8 r は非投機的であるので、これらは例外を延期させない。したがって、例外は発生され処理される。例えば、命令 I 4 r がページ・フォールドを発生したと仮定する。制御は、ページ・フォルトに対処する役割を担う例外ハンドラに移り、このフォルトを処理する。例えば、プログラムの実行を中断する場合があったり、あるいはメモリ・ページを仮想メモリ・スワップ・ファイルから読み込む場合もある。

#### 【0046】

先に注記したように、正しいプログラム状態を保存するために、違反した投機的依存チェーンからの命令のみが、リカバリ・コードの実行中プロセッサ状態を変更することを許される。図3に示す例では、命令 I 4、I 6、および I 8 のみが、命令 I 4 r、I 6 r、および I 8 r として再実行され、他の命令は再実行されない。この選択的再実行を行うには、最も早い命令から始まり、宛先がchk. s 命令によって読み出された命令で終わる、投機的依存チェーン内の命令全てのコピーを作る。このコピーを「リカバリ・コード」と呼び、chk. s 命令は、chk. s 命令がDETに遭遇した場合、制御をリカバリ・コードに移管する。リカバリ・コードの終了時に、コンパイラは命令 I 9 へのブランチ・バックを発行する。リカバリ・コードは、対応するchk. s 命令が実行されるときにのみ実行されるので、そしてchk. s は常に非投機的であるので、リカバリ・コード内の命令は全て非投機的である。したがって、リカバリ・コード内の命令は、非投機的バージョンに変換される（必要であれば）。図3に示す例では、命令 I 4、I 6、および I 8 のメインライン・バージョンには全て投機的とマークされており、一方リカバリ・コードのコピー（命令 I 4 r、I 6 r、および I 8 r

）には全て非投機的とマークされている。同じリカバリ・コードが、多数の `chk. s` 命令のターゲットとされる場合もあり得る。更に、別個の `chk. s` 命令に別個のリカバリ・コード・セグメントに分岐させることによって、同じ投機的依存チェーンに多数のリカバリ・コードが関連付けられることもある。

#### 【0047】

DETの存在は、例外状態が投機的依存チェーン内のある命令に発生したことを示す。したがって、いずれの命令を再実行する前にも、最初に、関連する例外ハンドラを活性化することによって例外状態を処理する。本発明の一実施形態は、この要件を自動的に満たす。何故なら、リカバリ・コードは、投機的依存チェーン内にある全ての関連する命令の非投機的コピーを含み、非投機的命令は直ちに例外を通報するからである。リカバリ・コードの実行時に、元の例外が違反命令によって再度発生され、適切な例外ハンドラが活性化される。例外ハンドラが例外状態を訂正した後、再度制御をリカバリ・コードに戻し、残りの命令を実行し続けてから、メインライン・コードに戻る。

#### 【0048】

本発明は、いずれの特定のDETフォーマットにも依存しない。好適な実施形態では、DETは単に延期した例外が存在することを示すだけで、これ以上の情報を含まない。代替実施形態には、特定の例外ハンドラが必要とし得るその他の情報、例えば、例外の種類、違反命令のアドレス等を含むように、DETを定義できるものもある。

#### 【0049】

また、本発明の別の態様は、データ投機のように、別の種類の投機からの回復を可能にするものもある。本発明の一実施形態では、格納命令に先立ち順不同で進められるロード命令を用いて、データ投機を例示し、図4ないし図6を参照しながら説明する。ここで用いる場合、ロードおよび格納命令を引用する際、命令が他の機能を実行するか否かにはかかわらず、それぞれリードおよびライトをメモリに対して実行するあらゆる命令を示すものとする。ロード命令の方が、メモリ・レイテンシのために、通常他の命令よりも必要な実行時間量が高い。ロード命令をプログラムの実行の初期に移動させることによって、コンピュータにおけ

る命令実行の効率が向上する。繰り上げロードと呼ばれるロードは、メモリの使用を必要とするアクティビティの実行並列性を高めることができる。

#### 【0050】

先に概述したように、コンパイラはどこでロード命令および格納命令が衝突する（即ち、共通メモリ位置をアクセスする）のか、100パーセントの確信度で検出することはできない。このために、ロード・レイテンシが重複しない、即ち、ロードをこれと衝突しそうな格納の前に移動しない、より控えめな命令スケジュールとならざるを得ないという点で、並列性の達成に対する障壁となる場合が多い。しかしながら、これらの場合の多くでは、ロードおよび格納命令は実際には衝突しない。したがって、本発明の一実施形態は、単一または多元プロセッサ・システムにおけるプログラム実行の並列性を改善する一手段として、ロード命令およびこれに依存する計算を、潜在的に衝突し得る格納命令の前に実行することを可能にする。

#### 【0051】

図4に示す単純な元のコード30について検討する。コード30は、レジスタr3の内容を、レジスタr1の内容によってインデックス（指標付け）されているメモリ位置に格納する命令I22、レジスタr2の内容によってインデックスされているメモリ位置の内容をレジスタr4にロードする命令I24、およびレジスタr4およびr6を加算し、その結果をレジスタr5に書き込む命令I26を含む。コンパイラがコード30をスケジュールする場合、命令I22およびI24を実行する際にレジスタr1の内容がレジスタr2の内容と同一であることは不可能ではないが、可能性は低いと判定すると仮定する。更に、コンパイラは、命令I22の前（またはこれと並列に）命令I24およびI26をスケジュールする方が一層効率的であると判断すると仮定する。命令の並列スケジュールリングに関して、単一プロセッサ・システムにおいても、単一プロセッサは典型的に多数の実行ユニットを含み、ここで多数の命令を並列に実行可能であることは認められよう。

#### 【0052】

図5は、スケジュール化コード40を示す。これは、本発明の一実施形態にし

たがって、コンパイラが図4の元のコード30をスケジュールしたときに生成されたものである。コード40は、命令I22（格納命令）の前にスケジュールされた命令I24およびI26を含む。「. a」（アドバンス命令を示す）がロード命令に添付されていることに注意されたい。これは、このロード命令が繰り上げロード・アドレス表（ALAT）内にロード・アドレスを記録していることを示す。命令I25は、ALATをチェックし、ロード（I24）および格納（I22）命令が同じメモリ位置をアクセスしたか否かについて判定するchk. a命令である。レジスタr1およびr2の内容が等しくなかった場合、これらの命令は同じメモリ位置をアクセスしなかったことになり、chk. a（I25）はなにもしない。しかしながら、レジスタr1およびr2の内容が等しかった場合、chk. a命令（I25）はデータ投機エラーを検出し、命令I24rから始まるリカバリ・コードに分岐する。命令I24rは、ロード命令を再実行し、格納命令（I22）の後にロード命令を再実行するので、適正な結果をレジスタr4にロードさせる。命令I26rは、加算命令を再実行し、正しい結果をレジスタr5に書き込み、命令I23rは再度命令I25に分岐し、データ投機エラーがないことを検証する。

#### 【0053】

図6は、図3および図5に示したスケジュール化コードの変化を発生するためにコンパイラによって実施可能なルーチンの一例を示すフローチャートである。他の実施態様も可能であるので、このフローチャートは単なる一例として提示するに過ぎない。本発明は、特定のプログラム言語やコンピュータ構成の使用に限定されるものではなく、単一または多元プロセッサ・システムに用いられるコンパイラに適用可能である。

#### 【0054】

図6のプロセスは、コンパイラによって未だスケジュールされていないソース・コンピュータ・プログラムにおける命令を表わす、従来の依存性グラフ（dependency graph）をコンパイラが作成するときに、ステップ503において開始する。本発明は、いずれの特定形式のグラフにも限定される訳ではない。依存性グラフは、ソース・コンピュータ・プログラムのセグメントを表

わす少なくとも1つのパス、およびコンピュータ・プログラムのセグメントにおける各命令を表わすノードを有する図を含み、いくつかの形態を取ることができる。各プログラム毎の依存性グラフは、典型的に、複数のパスを含み、その各々が複数のノードを有する。命令を表わすノードには、当該命令を実行するために必要なクロック・サイクル数のように、命令に関連する情報を注釈として付けるとよい。典型的に、図においてノードを接続するアーク (a r c) は、命令間の依存性を示す。

#### 【0055】

ステップ506において、コンパイラはグラフを検討し、グラフのどのパスが、開始から終了まで最も長い全実行時間を要することになる命令シーケンスを含むかについて判定を行う。

#### 【0056】

ステップ509において、コンパイラは、プログラムにおける最長パスの実行を最適化しようとする。何故なら、最長パスは、命令シーケンスの実行時間を制限する、プログラム内のクリティカル部分を表わすからである。コンパイラによって、従来の最適化技法を用いることができ、更に別の最適化技法も、繰り上げロードおよびそれに依存する計算に関係する毎に、以下で説明する。

#### 【0057】

前述のように、コンパイラが最長クリティカル・パスを最適化することができる1つの方法は、データ投機によることである。本発明の一実施形態では、これは、リード動作を含むロード命令のような命令を、ライト動作を含む格納（ストア）命令よりも前に、プログラムの実行の早期に移動させることを含む。ステップ512において、コンパイラは、ロード命令が最長クリティカル・パス内にあるか否かについて判定を行う。ロードが最長パス内にある場合、コンパイラは、最適化方法の1つとして、このロードおよびそれに依存する命令を繰り上げる。これについて以下で説明する。

#### 【0058】

短縮すべきパス内でロード命令が発見された場合、コンパイラは、次に、ステップ521においてどの計算命令が、ロード命令によって読み取られたデータに

依存するののかについて判定を行う。計算が依存性であるのは、これらがロード命令の完了から得られる値の使用を必要とする場合である。

#### 【0059】

ステップ524において、ロード命令を、スケジュール化命令シーケンス内のその場所から除去する。ステップ527において、ロードに依存する計算（ステップ521において識別した）を繰り上げ、ロード命令に続けることにより、ロード命令およびこのロード命令に依存する計算双方を繰り上げて、命令シーケンスの最適化を図る。コンパイラは、「ld. a」と称するロード命令を、その実行によってプログラムの性能が全体的に向上し得るような位置の前に繰り上げる。

#### 【0060】

前述したように、格納命令は、依存性グラフのパスにおいて、ロード命令「ld. a」の前に存在する可能性があり、コンパイラはロードおよび格納が衝突するか（即ち、同じメモリ位置を使用するか）否かについて判定できない場合がある。ステップ530において、コンパイラは、ロードおよび格納が衝突しない絶対的な確信度があるか否かについて判定を行う。

#### 【0061】

移動させる前に、ロードおよび格納が衝突しないことの確信が判定できない場合、ステップ533において、ステップ524において除去したロード命令を、図3および図4に関連付けて先に説明したchk. a命令のようなチェック命令と置換する。chk. a命令は、ロード命令と置き代わり、ロードを繰り上げなければスケジュールされていた場所で実行される（以下で説明する）。

#### 【0062】

ステップ536において、コンパイラは、ステップ527において繰り上げた、繰り上げロードおよびこの繰り上げロードに依存する計算に対して、リカバリ・コードを発生する。リカバリ・コードは、必要であれば、以下で述べるようにchk. a命令によってコールされる。

#### 【0063】

ステップ512においてロード命令が最長クリティカル・パス内で発見されな

い場合、またはステップ530において、ロードが、衝突する格納命令の前に繰り上げられていないことをコンパイラが絶対的に確信すると判定した場合（したがって、チェック命令やリカバリ・コードは不要である）、またはリカバリ・コードをステップ536において発生した場合、プロセスはステップ539に進み、ここでコンパイラは、潜在的に最適化が可能な最長クリティカル・パスが残っているか否かについて判定を行う。コンパイラは、当該コンパイラが可能な限りのソース・プログラムを最適化するまで、プログラムにおける次の最長パスの各々を最適化して行くことによって、ソース・プログラムの実行並列性を向上させることができる。

#### 【0064】

ステップ539において、コンパイラはもはやプログラムを最適化できないと判定した場合、プロセスはステップ542に進み、コンパイラは、最適化した命令シーケンスを実行のためにスケジュールする。しかしながら、ステップ539において、潜在的に最適化が可能な最長クリティカル・パスが未だ残っているとコンパイラが判断した場合、コンパイラはステップ506において次の最長パスを特定する。このように、最適化が可能なグラフ内のパス全てを最適化するまで、プロセスは継続する。

#### 【0065】

ステップ542において、コンパイラは、命令の実行をスケジュールし、前述の最適化手順によって行われた実行順序のあらゆる変更をも反映する。コンパイラは、プログラムにおける命令実行のスケジュールを、多数の方法で行うことができ、並列実行ユニットを利用することができるが、本発明はいずれの特定のスケジューリング機構にも限定されない。図4において先に説明した例では、最適化によって得られたコードを図5に示し、リカバリ・コードを命令I24r、I26rおよびI23rとして記す。

#### 【0066】

図7は、図6と関連付けて先に説明したような技法によって最適化した命令シーケンスを実行する際に、コンピュータ・システムが実行するルーチンを示すフローチャートであり、ロード命令および依存する計算を、順不同に格納の前に進

めることを含む。

#### 【0067】

スケジュール化した命令シーケンスの実行は、ステップ603において開始する。命令シーケンスの実行中、繰り上げロード（例えば、図5におけるld. a）をステップ606において実行する。繰り上げロード命令を実行した後、ステップ609において、ALATを更新し、繰り上げロード命令によって読み取られたメモリ位置（例えば、r2内のアドレス）の範囲を記録する。ステップ618において、実行した格納動作のように、後に実行される格納命令が、繰り上げロードおよび格納命令が共通のメモリ位置にアクセスしたか否かについて判定することを可能とするために、ALAT内にエントリを作る。特定の繰り上げロードおよびそれに対応する格納命令のメモリ・アドレスの範囲を比較することを可能にするあらゆる構造が使用可能であるので、本発明は、いずれの特定のALAT構造にも限定される訳ではない。

#### 【0068】

図8に示す一実施形態では、ALAT内のエントリは、物理メモリ・アドレス・フィールドおよびメモリ・アクセス・サイズ・フィールドを含み、これらが一緒になってアクセスされるメモリ位置の範囲を規定する。本発明は、このメモリ位置範囲を規定する方法に限定される訳ではなく、多数のその他の技法も採用可能である。例えば、アクセスされるメモリ範囲は、開始および終了メモリ・アドレスによって、またはメモリ終了アドレスおよび範囲によって特定することも可能である。図8に示す実施形態では、ALATは有効ビットのフィールドも含み、エントリが有効か否かについて示すために用いられる。以下で説明するが、本発明の一実施形態では、ALAT内のエントリを無効化することが望ましい時点がある（例えば、2つのアプリケーション間のコンテキスト切り替え）。有効ビットは、かかる無効化を実行する際に便利な機構を提供する。

#### 【0069】

本発明の一実施形態では、各々対応する格納よりも前に繰り上げられた多数のロードが同時にある場合もあり得る。以下で説明するが、プログラムの実行中、各ロードおよび格納対間に衝突が起きなかったことを検証する技法が備えられて

いる。したがって、本発明の一実施形態では、ALATは、対応する格納命令との可能な衝突を判定するために、エントリを特定することができるよう、対応する繰り上げロード命令を一意に識別する情報を含む。この一意の識別は、多くの方法で実現可能であり、本発明はいずれの特定な実施態様にも限定されるものではない。図8に示す実施形態では、特定の繰り上げロードに対するエントリに、繰り上げロード命令において用いられるレジスタ番号およびレジスタの型（汎用または浮動小数点）に基づいてインデックスを付ける。各命令毎に用いられるレジスタ番号は、コンパイラによって割り当てられ、各繰り上げロード命令毎に、確実に唯一のレジスタが使用されるようにする。

#### 【0070】

ステップ606（図7）において繰り上げロードを実行し、ALAT内にエントリを作る前に、ターゲット・レジスタ番号をインデックスとして用いてALATにアクセスし、ALATをチェックして、繰り上げロードが用いるターゲット・レジスタ番号に対応するエントリが既に存在するか否かについて判定を行う。エントリがある場合、これを除去する。何故なら、現在の繰り上げロードには関係ない情報を含んでおり、以前の繰り上げロードの実行中に入力された可能性が非常に高いからである。以前に実行した繰り上げロードからのデータを、既存のエントリからクリアした後、または現在の繰り上げロードのターゲット・レジスタ番号に対応する空のエントリをALAT内で発見した場合、ターゲット・レジスタ番号でインデックスした新たなエントリを、現在の繰り上げロード命令に対して作成する。

#### 【0071】

繰り上げロードを実行しALATを更新した後、ステップ612において、繰り上げロード命令の結果に依存する計算命令（例えば、I24、I26）を実行する。前述した例では、依存計算は、図5に示すように、加算命令I26を含む。繰り上げロード命令に続く他の命令や、ロードを繰り上げた後の格納命令に先立つ他の命令は全て、ステップ615で実行される。

#### 【0072】

ステップ618において、ロードを繰り上げ、このロードと衝突する可能性が

ある格納命令（例えば、I 2 2）を実行する。図5に示す例では、格納命令I 2 4は、r 1内のアドレスにアクセスする。格納命令I 2 2を実行する際、当該格納によって書き込まれるメモリの物理アドレスおよび領域のサイズを用いて、A L A T内にある有効なエントリ全てを探索する。この探索は多数の方法のいずれでも行うことができ、本発明はいずれの特定の技法にも限定されるものではない。本発明の一実施形態では、A L A Tは、完全連想内容参照可能メモリとして構成されているので、有効なエントリは全て同時に探索される。エントリを探索して、格納命令といずれかの繰り上げロード命令との間に衝突が発生したか否かについて判定を行う。繰り上げロード（例えば、I 2 4）のメモリ空間の範囲が、格納命令（例えば、I 2 2）のメモリ空間の範囲と重複するA L A T内で発見された場合、衝突が発生したことになる。本発明の一実施形態では、衝突を検出した場合、衝突した繰り上げロードに対応するアドレスに対するA L A T内のエントリをステップ6 2 1において除去し、衝突を示す。ステップ6 2 1において、格納命令（例えば、I 2 2）によってアクセスされたメモリ空間が、いずれの繰り上げロード命令（例えば、I 2 4）のそれとも重複していない場合、個々の繰り上げロード命令に対応するA L A T内のエントリは、A L A T内に残る。

#### 【0073】

繰り上げられたロード命令は、各々このロード命令と衝突する潜在的な可能性がある、多数の格納命令の前に移動させてもよいことは認められよう。一連の格納命令に続く単一のチェック命令を用いて、多数の格納命令のいずれかがロード命令と衝突したか否か検出することも可能である。何故なら、各格納命令の実行は、A L A Tにおけるエントリ全てを探索し、衝突が発生したか否かについて判定することを含むからである。このように、チェック命令は、プログラムにおける格納命令の数には無関係である。何故なら、別個のチェック命令は、ステップ5 3 3において繰り上げた各ロード命令と置き代わり、各チェック命令は、ステップ6 2 3において、以下で説明するように、A L A Tを見直すからである。

#### 【0074】

また、繰り上げロードおよび格納命令によってアクセスされたデータの開始アドレスが同一でなくても、前述したように、これらの命令の間で衝突は発生し得

ることも認められよう。即ち、各命令は、多数のデータ・バイトにアクセスする場合があるのである。したがって、繰り上げロードによって読み取られたデータが占めていたメモリ・アドレスの範囲と、格納によって書き込まれたデータが占めていたメモリ・アドレスの範囲との間に少しでも重複があれば、衝突が発生する可能性がある。衝突の検出は多数の方法で実行することができ、本発明は、いずれの特定の実施態様にも限定されるものではない。例えば、繰り上げロードによって書き込まれたデータおよび格納によって読み出されたデータのアドレス間で、範囲全域の比較を行うことができる。しかしながら、範囲全域の比較は、ハードウェアで実現するには費用がかかる可能性がある。したがって、本発明の一実施形態によれば、繰り上げロードおよび格納によってアクセスされるデータのアドレスの範囲全域の比較を行わずに、衝突を判定する技法を採用する。

#### 【0075】

本発明のこの実施形態によれば、メモリに格納されるデータのサイズ整合を優先し、データ・ブロックの開始アドレスをそのサイズの偶数の倍数とすることが好ましい。例えば、4バイトを含むデータ・ブロックは、下位2ビット(LSB)がゼロであるアドレスに格納することが好ましく、8バイトのブロックは、3 LSBがゼロであるアドレスに格納することが好ましい等である。データをサイズ整合すると、繰り上げロードおよび格納によってアクセスされるデータの開始アドレスの直接的な同等性比較を単に実行するだけで、衝突を検出することができる。直接同等性比較は、範囲全域の比較よりも、ハードウェアで実現する場合、はるかに低いコストで済むことは認められよう。データが不整合の場合、不整合の制限量があり、不整合データがより大きなサイズの整合データ範囲に納まることができるようにするのであれば、本発明の一実施形態では、ハードウェアは、あたかもより大きなサイズの整合データ範囲にアクセスしているかのように、不整合データにアクセスする命令(例えば、ロード)を処理する。例えば、ロードが、メモリ内の8バイトの不整合データにアクセスする場合、ロードによってアクセスされるデータが32バイトのサイズ整合範囲には納まるのであれば、32バイト・データにアクセスするものとして、このロードを扱う。命令をより大きなサイズ整合データ・ブロックを用いるものとして扱うことにより、繰り上げ

ロードおよび格納命令によってアクセスされるデータ・アドレスの重複が生ずるという状況があり得るが、実際にはこれらの命令の一方によってアクセスされているだけであり、実際のデータ重複（前述の例では8バイト）ではないので、偽りの衝突が検出されることになる。この性能上の欠陥は、衝突検出のためのハードウェア複雑化の低減に対して支払う代償である。データの不整合が著しく、正当な大きなサイズ整合データ範囲には納まらない場合、ハードウェアはロード命令も格納命令も処理しない。ロードについては、ALATにエントリを挿入せず、そのために衝突が指示される。この結果、偽りの衝突が検出されるが、この性能上の欠陥は、衝突検出のためのハードウェア複雑化の低減に対して支払う代償である。著しく不整合の格納命令については、命令を一連の小さな格納命令に分離することも可能である。一実施形態では、大きな格納を一連の多数の格納に分離するハードウェアを用いることができる。別の実施形態では、サイズ整合データ範囲に納まることができない不整合格納は、割込を発生することができ、オペレーティング・システムが、この格納命令を一連の小さな格納命令に分離することによって、この格納を処理する。不整合格納命令を処理する双方の実施形態において、ステップ618に記述するように、一連の小さな格納の各々を、ALAT内の有効なエントリに対してチェックする。小さな格納のいずれかのメモリ空間範囲が、繰り上げロードのメモリ空間範囲と重複する場合、ステップ621に記述するように、衝突が指示されよう。このように、小さな格納の各々を実行することによって、単一の大きな格納を実行する場合と同じ結果が得られる。

#### 【0076】

本発明の別の実施形態では、アドレスの1つ以上の最上位ビット(MSB)を無視し、同等性比較に部分アドレスを用いることによって、衝突検出に用いられるハードウェアの更なる削減を達成する。1つ以上のMSBを無視することによって、ALATのサイズ縮小だけでなく、同等性比較を実行するハードウェアの削減ももたらされる。何故なら、各エントリ毎にALATに格納するビット数が減少し、比較するビットも減少するからである。例えば、64ビット・データ・アドレスでは、ロードの下位20ビット(LSB)のみをALATにセーブし、同等性比較に用いることができる。

## 【0077】

1つ以上のMSBを無視すると、その結果何らかの偽りの衝突が検出される可能性があることは認められよう。即ち、格納命令を実行する際に（例えば、ステップ618において）ALATを探索すると、同等性比較を実行するLSB（例えば、20LSB）については、格納のデータおよびロードのデータの完全な開始アドレスが一致するが、無視した1つ以上のMSBについては異なるという場合がある。このことが発生した場合、衝突が実際に発生したかのように、例えば、制御のフローをリカバリ・コードに切り替えることによって、図7のルーチンを実行する。したがって、偽りの検出は、実際には発生しなかった衝突からの回復による何らかの性能上の不利益を被る結果となることは認められよう。この性能上の不利益は、衝突検出のためのハードウェアの複雑化を低減することに対して支払われる代償である。検出方式において（少しでも無視する場合には）どれ位MSBを無視するかについて決定する際、これら相反する要因間のバランスを考慮することができる。

## 【0078】

ステップ623において、ステップ606で実行した繰り上げロード命令に対してchk. a命令（例えば、I25）を実行する。一実施形態では、chk. a命令は、ALATを見直し、繰り上げロード命令（例えば、I24）のエントリがあるか否かについて判定を行うことにより、衝突が発生したか否か判定する。インデックスとして、ステップ621においてALAT内で更新した情報に対して、繰り上げロード命令（例えば、I24）が用いたターゲット・レジスタのアイデンティティ（同一性）およびレジスタ・タイプを用いて、chk. a命令はALATを見直す。ステップ624において、chk. a命令によって置換された特定の繰り上げロードに対応するエントリがALAT内で発見された場合、chk. a命令は、格納命令（例えば、I22）およびこの格納よりも上に繰り上げられたロード命令（例えば、I24）が衝突しなかったことを認識する。したがって、格納命令によって読み取られたデータは有効であり、ルーチンは、ステップ630に進み、命令シーケンスの実行を終了する。

## 【0079】

しかしながら、ステップ624においてchk. a命令がALATを見直し、繰り上げロード命令（例えば、I24）が用いたレジスタ・アドレスに対応するエントリをALAT内において見出せなかった場合、chk. a命令（例えば、I25）は、格納および繰り上げロード命令は同じメモリ空間にアクセスした（即ち、衝突した）と判断する。したがって、繰り上げロード命令およびこの繰り上げロード命令（例えば、I24）に基づいて実行した計算命令の精度を確認するために、更に別のステップを実行する。一実施形態では、可能な衝突を検出した場合、プログラムの制御フローを変更し、リカバリ・コードを実行する。前述したように、これは多数の方法で行うことができる（例えば、例外処理技法に分岐する、即ち、これを用いることによる等）。ALATは、実行するプログラムに含まれ得る繰り上げロード命令の全てに対応するには不十分なエントリ数でも実現可能であることは認められよう。図7に示す実施形態では、chk. a命令は、ステップ633（図7）においてリカバリ・コードに分岐する。リカバリ・コードの一例は、図5において、命令I24r、I26rおよびI23rとして示す。これらは、本質的に、コメントI24、I26およびI23のコピーである。

#### 【0080】

ステップ636において、ステップ524で繰り上げたロード命令（例えば、I24）を再実行する。ステップ639において、繰り上げロード命令（例えば、I24）に依存する命令を再実行する。図5の例では、再実行されるロード命令I24rおよび依存する加算命令I26rを示す。これらの命令を再実行するのは、ロード命令I24rおよびそれに依存する計算I26rの適正な結果を得る格納命令の後である。一実施形態では、リカバリ・コードは、当初実行したロードおよび依存する計算命令と同じ結果が得られると、コンパイラが判断した命令の組み合わせであれば、いずれでもよい。

#### 【0081】

ステップ642において、制御フローはリカバリ・コードからコンパイルした実行スケジュールに戻る。これは、例えば、図5におけるI23rのような分岐命令を用いて行うことができる。次に、ステップ630において、プログラムの

終端まで、スケジュールされた命令の実行を継続する。

#### 【0082】

例えば、一実施形態では、ALATは、32個のアドバンス命令に対応するエントリの空間を有する。実行するプログラムの中に32個よりも多いアドバンス命令がある場合、ALATは、これらアドバンス命令の全てに関する情報のために十分な空間を有さないことになる。ALATが満杯となり、新たなアドバンス命令が実行された場合、入れ換え方式を用いて、ALAT内の有効なエントリを取り出し、新たな命令のために余裕を作ることができる。更に、本発明の一実施形態では、実行時にプロセス間（例えば、別個にコンパイルしたプログラム間）で切り替えを行う場合、ALAT内のエントリを、後に復帰させるためにセーブすることができ、あるいは無効化することも可能である。このように、場合によっては、chk. a命令は、個々のアドバンス命令に対して、その命令に衝突が発生していなくても、エントリを見つけることができない場合もある。

#### 【0083】

前述したように、本発明は特定のALAT構造に限定される訳ではなく、ロードおよび格納命令間に衝突があったか否かについて判定を行うために、他の代替実施形態を含むことも可能である。例えば、他のデータ構造または比較回路を用いてもよい。また、用いるALATまたはその他の構造は、用いるサイズやフィールド数が異なってもよい。加えて、多数のレジスタ集合の各々に、別個のALATまたはデータ構造を用いてもよい。例えば、一実施形態では、ALATは汎用および浮動小数点レジスタ集合に用いることができる。

#### 【0084】

以上延期例外処理およびデータ投機を参照しながら本発明について説明したが、これに限定される訳ではない。一般に、本発明は、投機的に実行されるあらゆるタイプの命令セグメント、投機的に実行された命令の実行完全性の検証、および検出したあらゆる問題を訂正するためのリカバリ・コードの実行を包含するものである。本発明は、制御投機的およびデータ投機的双方である命令を含むように拡張することも可能である。

#### 【0085】

chk. sおよびchk. a命令からリカバリ・コードへの制御移管は、多数の方法のいずれでも実現可能である。例えば、chk. sおよびchk. a命令は、各々、分岐命令として振る舞うことができ、リカバリ・コード内の最初の命令のアドレスを、chk. sまたはchk. a命令自体に含ませる（図3に示すように）。あるいは、chk. sまたはchk. a命令は、特定の例外を発生することができ、例外ハンドラがchk. sまたはchk. a命令内の値を用いて、対応するリカバリ・コードを特定し、制御をこのリカバリ・コードに移管することも可能である。また、例外ハンドラは、chk. aまたはckh. s命令のアドレス、即ち、命令が格納されているメモリのアドレス位置を用いて、リカバリ・コードの位置を特定することも可能である。リカバリ・コードは、コンパイラが作成する表に基づくことができる。この表は、コンパイラによってコンパイルされたソース・プログラムに追加されたチェック命令のアドレスを含む。したがって、実行されるリカバリ・コードは、どのチェック命令を実行するかによって特定される。

#### 【0086】

本発明は、コンパイラが、繰り上げた命令が後の命令と衝突しないことに確信がない場合でも、命令を順不同に繰り上げることを可能にする。前述したように、従来のコンパイラの中には、ロードおよび格納が衝突しないことに確信がなくても、単一のロード命令を格納よりも前に移動させ得るものもある。実行時に、衝突があった場合、コンパイルした実行スケジュールにおいてロード命令をインラインで再実行する。対照的に、本発明の一実施形態では、単にロードを格納の前に繰り上げるだけでなく、それに依存する計算も繰り上げることによって、命令実行の最適化を図る。これによって、コンパイラおよびスケジューラは、多数の実行ユニットと一度に最も効率的に使用することが可能となる。更に、衝突がある場合にロード命令を単に再実行する代わりに、チェック命令を実行し、衝突があったか否か判定を行い、ロード命令およびそれに依存する計算を含むリカバリ・コードに制御フローを変更する。このように、多数のコード・セクションを独立してしかも並列に実行することができる。

#### 【0087】

本発明は、c h k 命令、投機的依存チェーン、およびリカバリ・コード間の連携に関するコンパイラの部分の柔軟性を高めることができる。ここに含む例は比較的単純であるが、図5に示した例におけるように、投機的依存チェーンが単一の線形的な命令シーケンスで構成されているのではなく、多数のシーケンスを含む場合、または2つ以上の投機的依存チェーンが互いに依存しあう場合等、かなり複雑なコード構成も可能である。本発明は、これら種々の構成のアドレッシングにおいて高い柔軟性を持たせることによって、静的投機の理解が高まるに連れて、リカバリ・コードの使用において今後の改良を可能としている。

#### 【0088】

本発明は、c h k 命令の数および構成に関して、広い度合いの柔軟性を持たせている。例えば、単一のc h k. s は、投機的依存チェーンに沿った命令のいずれか1つの宛先を読み取るように構成することができ、または多数のc h k. s 命令を発行し、各々が異なる宛先を読み取ることも可能である。各c h k. s 命令も、同一または異なるリカバリ・コード命令集合を呼び出すことができる。

#### 【0089】

また、本発明は、D E T の存在を検出する代替実施形態も包含するものである。例えば、一実施形態では、明示的なc h k. s 命令がない。代わりに、非投機的命令の通常の実行の一部として、各非投機的命令によってD E T を検出する。この実施形態では、非投機的命令がD E T に遭遇した場合、例外を発生し、例外延期に対処する。別の例示的な実施形態では、各命令の宛先の代わりに、D E T を専用レジスタまたはメモリに格納する。

#### 【0090】

別の実施形態では、非投機的リカバリ・コードは、投機的インライン・コードと同じコードとすることも可能である。例えば、命令内に含まれる投機フラグに基づいて、各命令に投機的または非投機的とマークすることができるアーキテクチャも採用可能である。例えば、コンパイラは、命令のセグメントを投機的としてスケジューリングことができ、これらの命令を実行した後にD E T を検出することによって、延期例外ハンドラを活性化することができる。延期例外ハンドラは、単に投機的命令の投機フラグを切り替え、これらを非投機的命令に変換し、命

令を再実行し、以前に延期した例外を処理し、再度フラグを切り替えて命令を投機的命令に逆変換することができる。この実施形態はコンパイラのリカバリ・コードのスケジューリングにおける柔軟性を損なうが、コードによって使われるメモリ量の大幅な削減をもたらすことができる。加えて、投機フラグは、キャッシュ・メモリ内で切り替えればよいので、投機フラグを切り替えるのに要する時間も極力短くすることができる。

#### 【0091】

同様の実施形態では、投機的命令を非投機的として実行すべきコード・セグメントを識別するために1組のレジスタを定義することができる。この実施形態では、命令の投機フラグを切り替えて非投機的に実行する代わりに、非投機的に実行する命令を特定するインデックスをレジスタにロードすることを除いて、ほぼ前述と同様に機能する。

#### 【0092】

本発明の範囲内において、図面に示し本明細書に記載した実施形態には、種々の変更や修正も可能であることは理解されよう。前述の説明に含まれ添付図面に示した事柄全ては、限定的な意味ではなく、例示的な意味で解釈することを意図するものである。本発明は、請求の範囲およびその均等物における規定にのみ限定されるものである。

#### 【図面の簡単な説明】

##### 【図1】

本発明の実施形態を実施可能な汎用コンピュータのブロック図である。

##### 【図2】

3つの基本ブロックを含む元のコード・シーケンスを示す図である。

##### 【図3】

本発明の一実施形態による静的投機を用いて、図2の元のコード・シーケンスをスケジューリングすることから得られる、スケジュール化コード・シーケンスを示す図である。

##### 【図4】

メモリ・ロード命令に続いてメモリ格納命令を含む元のコード・シーケンスを

示す図である。

【図5】

本発明の一実施形態にしたがって、静的データ投機を用い、図4の元のコード・シーケンスをスケジューリングし、ロード命令を繰り上げた結果得られた、スケジュール化コード・シーケンスを示す図である。

【図6】

本発明の一実施形態にしたがって、格納命令の前に、ロード命令およびこれに依存する命令を繰り上げるプロセスのフローチャートである。

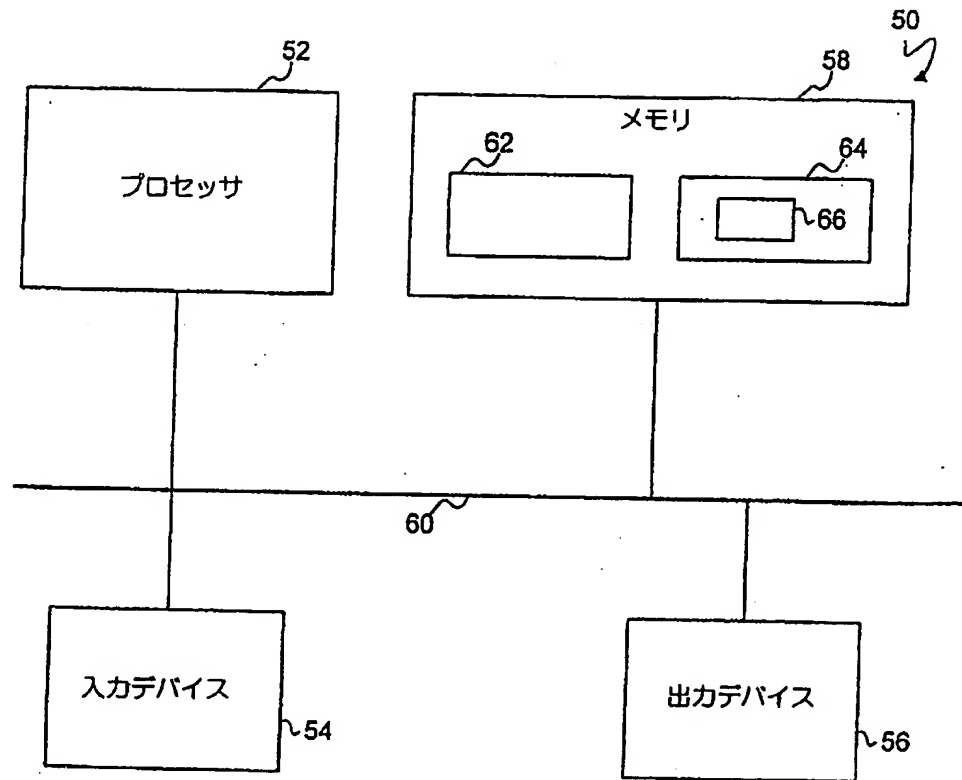
【図7】

本発明の一実施形態にしたがって、実行時に繰り上げロード命令を実行するプロセスのフローチャートを示す。

【図8】

本発明の一実施形態による繰り上げロード・アドレス表の一例を示す図である。

【図1】



【 FIGURE 1 】

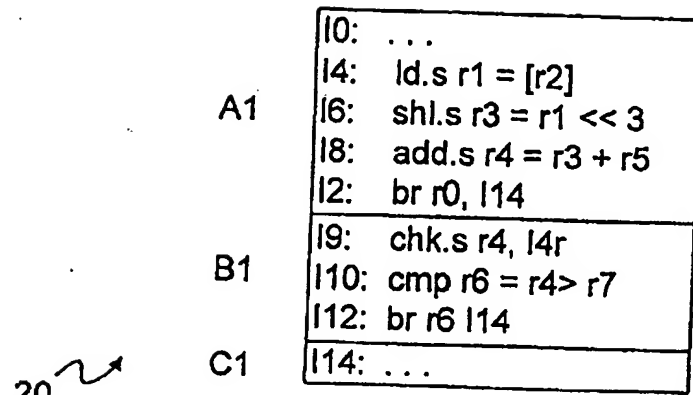
【図2】

	10:	...
A1	12:	br r0, l14
	14:	ld r1 = [r2]
	16:	shl r3 = r1 <<3
B1	18:	add r4 = r3 + r5
	110:	cmp r6 = r4 > r7
	112:	br r6, l100
C1	114:	...

【 FIGURE 2 】

元のコード

【図3】

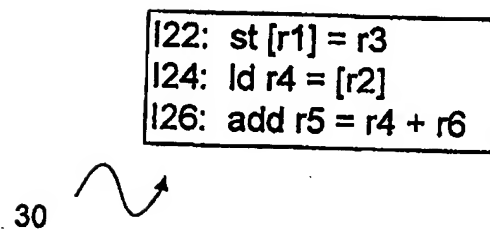


...  
l4r: ld r1 = [r2]  
l6r: shl r3 = r1 << 3  
l8r: add r4 = r3 + r5  
l9r: br l9

【 FIGURE 3 】

ライン外にリカバリを有する  
スケジュール化コード

【図4】



【 FIGURE 4 】

元のコード

【図5】

l24: ld.a r4 = [r2]
l26: add r5 = r4 + r6
l22: st [r1] = r3
l25: chk.a r4, l24r
l24r: ld r4 = [r2]
l26r: add r5 = r4 + r6
l23r: br l25

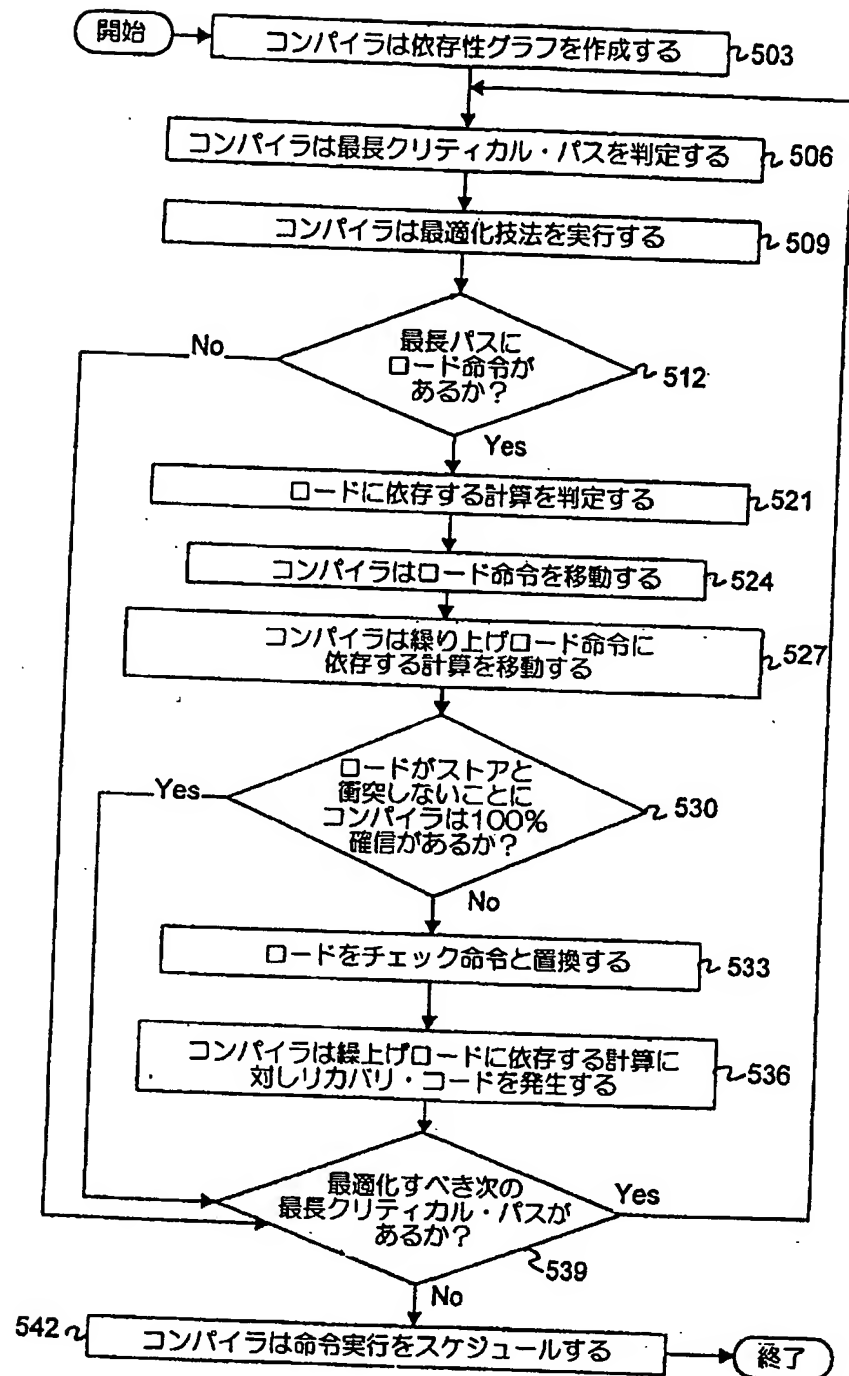
40

## 【 FIGURE 5 】

ライン外にリカバリを有する  
スケジュール化コード

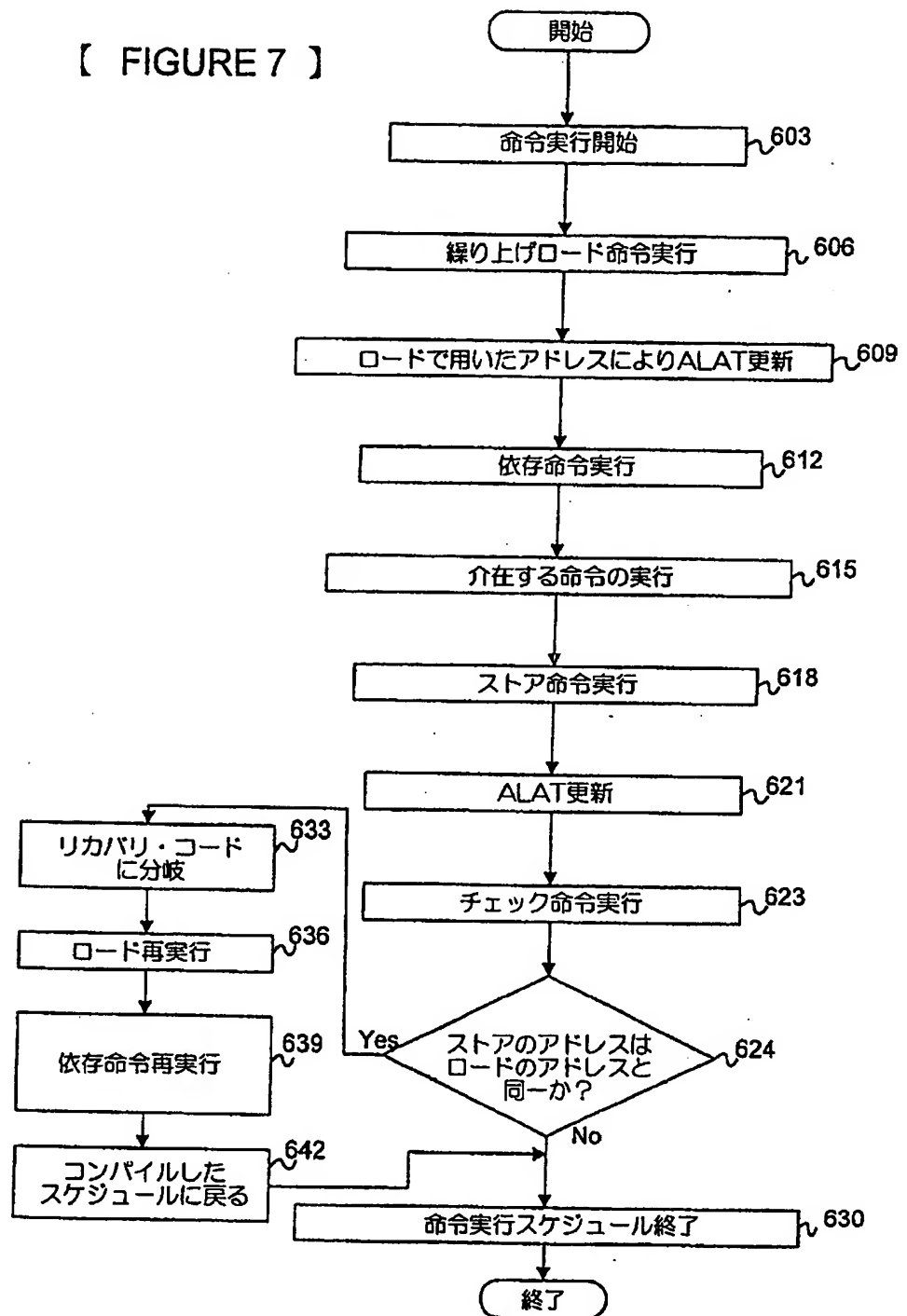
【図6】

【 FIGURE 6 】



【図7】

【 FIGURE 7 】



【図8】

繰り上げロード・アドレス・テーブル (ALAT)				
メモリ・ アドレス	メモリ・ アクセス・ サイズ	レジスタ 番号	レジスタ・ タイプ	有効ビット

【 FIGURE 8 】

【手続補正書】

【提出日】平成12年12月28日(2000.12.28)

【手続補正1】

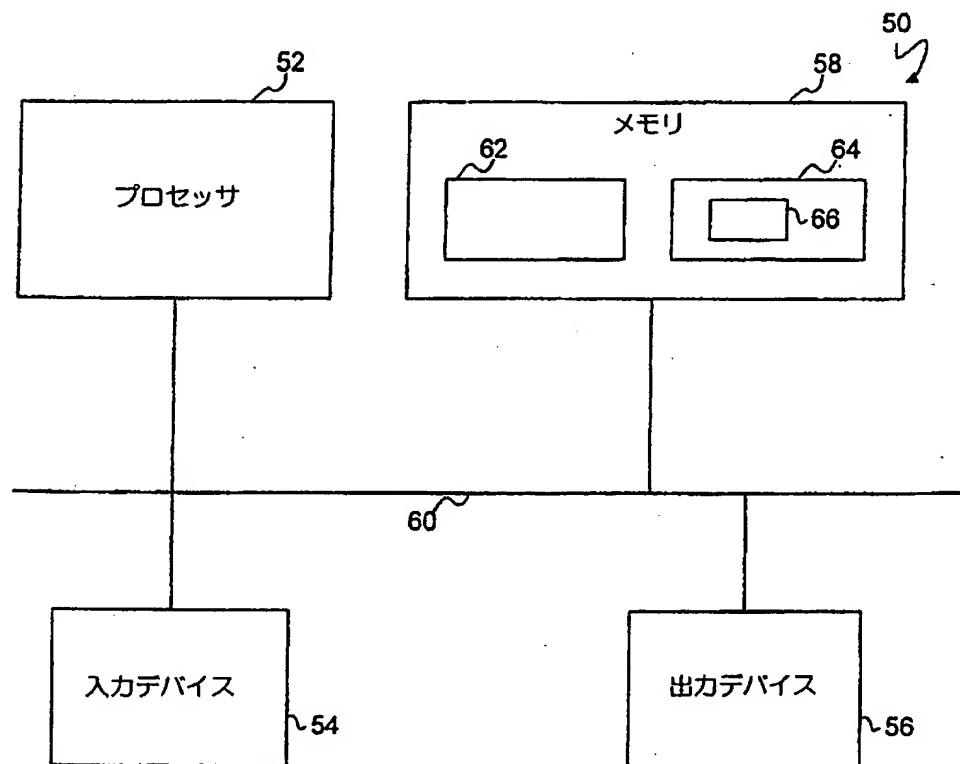
【補正対象書類名】図面

【補正対象項目名】全図

【補正方法】変更

【補正内容】

【図1】



【図2】

	10:	...
A1	12:	br r0, l14
	14:	ld r1 = [r2]
	16:	shl r3 = r1 << 3
B1	18:	add r4 = r3 + r5
	110:	cmp r6 = r4 > r7
	112:	br r6, l100
C1	114:	...

10 ~

元のコード

【図3】

	10:	...
A1	14:	ld.s r1 = [r2]
	16:	shl.s r3 = r1 << 3
	18:	add.s r4 = r3 + r5
	12:	br r0, l14
B1	19:	chk.s r4, l4r
	110:	cmp r6 = r4 > r7
	112:	br r6 l14
C1	114:	...

20 ~

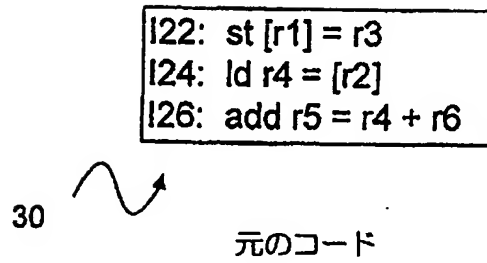
```

...
14r: ld r1 = [r2]
16r: shl r3 = r1 << 3
18r: add r4 = r3 + r5
19r: br l9

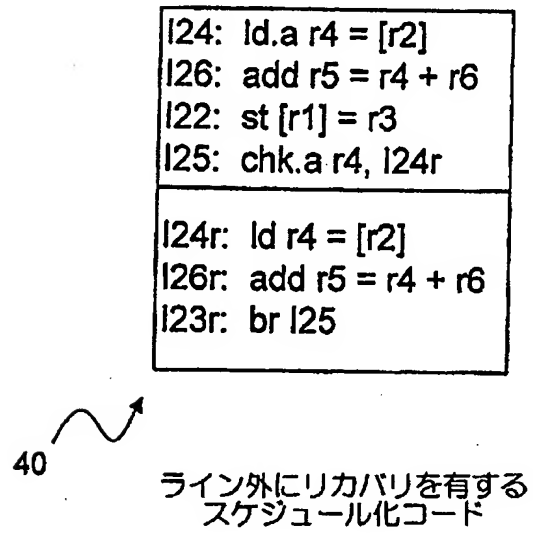
```

ライン外にリカバリを有する  
スケジュール化コード

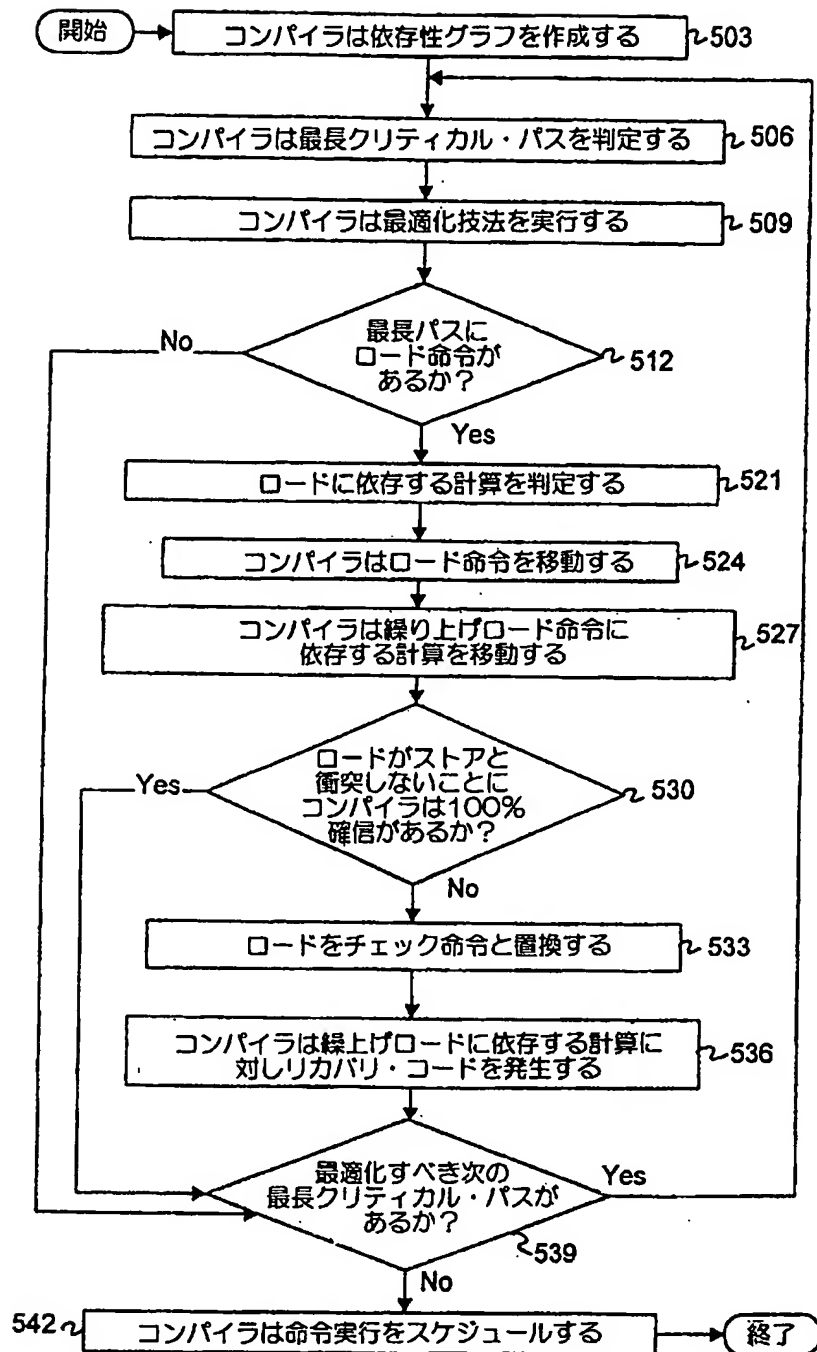
【図4】



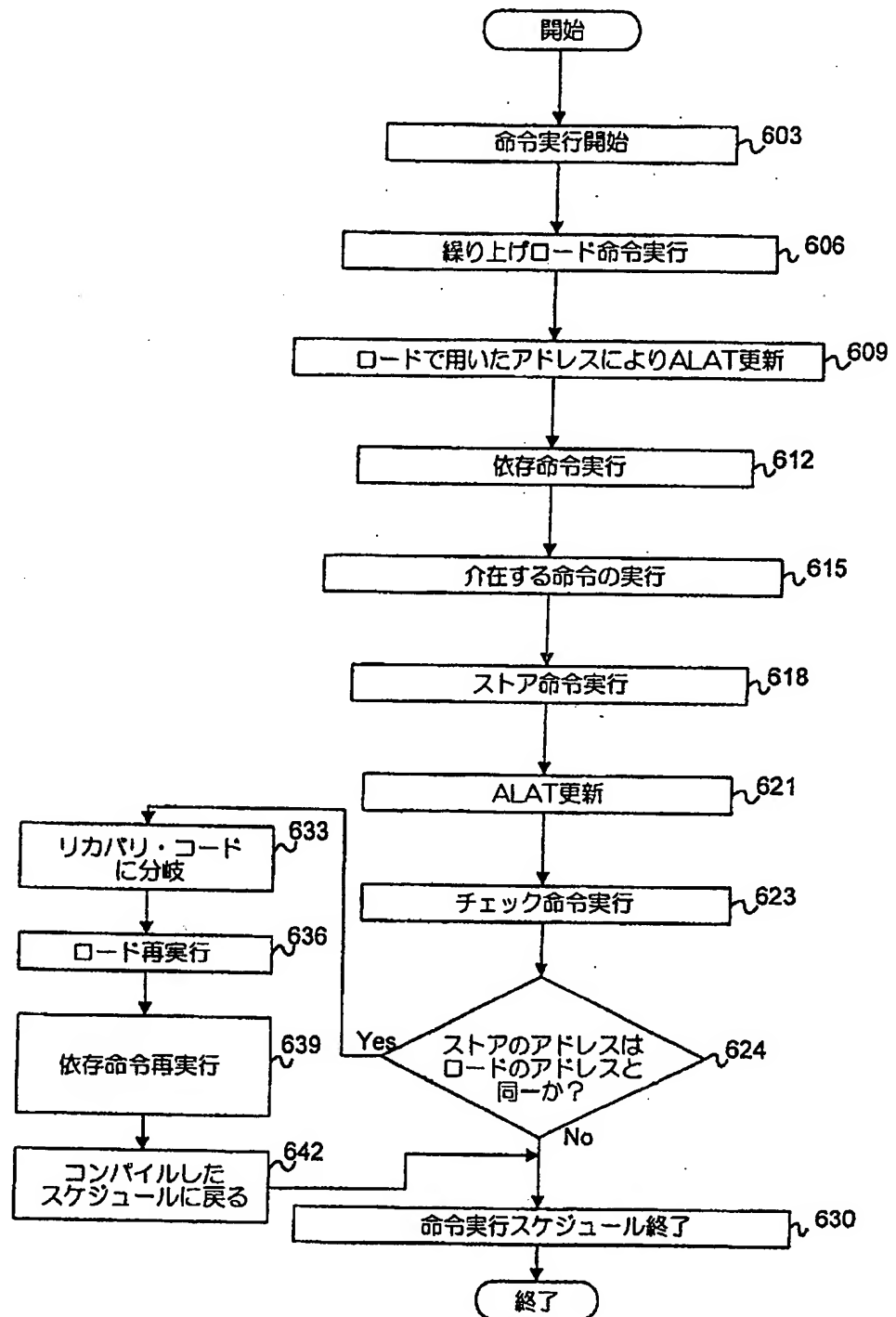
【図5】



【図6】



【図7】



【図8】

繰り上げロード・アドレス・テーブル (ALAT)				
メモリ・ アドレス	メモリ・ アクセス・ サイズ	レジスタ 番号	レジスタ・ タイプ	有効ビット

## 【国際調査報告】

## INTERNATIONAL SEARCH REPORT

International application No.  
PCT/US98/21465

## A. CLASSIFICATION OF SUBJECT MATTER

IPC(6) : G06F 9/38, 9/45, 11/28

US CL : 395/591, 706, 800.23, 800.24

According to International Patent Classification (IPC) or to both national classification and IPC

## B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 395/591, 706, 800.23, 800.24

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

APS

## C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 5,625,835 A (EBCIOGLU ET AL.) 29 APRIL 1997, SEE ENTIRE DOCUMENT.	1-33 AND 44-69
XP	US 5,778,219 A (AMERSON ET AL.) 07 JULY 1998, SEE ENTIRE DOCUMENT.	34-43

☐ Further documents are listed in the continuation of Box C.
 ☐ See patent family annex.

## \* Special categories of cited documents

"A" document defining the general state of the art which is not considered to be of particular relevance

"B" earlier document published on or after the international filing date

"L" documents which may show doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reasons (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T"

later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X"

document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y"

document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"Z"

document member of the same patent family

Date of the actual completion of the international search

15 JANUARY 1999

Date of mailing of the international search report

02 APR 1999

Name and mailing address of the ISA/US  
Commissioner of Patents and Trademarks  
Box PCT  
Washington, D.C. 20231

Facsimile No. (703) 305-1230

Authorized officer

WILLIAM M. TREAT

Telephone No. (703) 305-9699

Form PCT/ISA/210 (second sheet)(July 1992) \*

## フロントページの続き

(81)指定国 EP(AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OA(BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG), AP(GH, GM, KE, LS, MW, SD, SZ, UG, ZW), EA(AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZW

(72)発明者 ミルズ, ジャック・ディー  
アメリカ合衆国カリフォルニア州95124,  
サン・ホセ, シェバリエ・ドライブ 1768

(72)発明者 チェン, ウィリアム・ワイ  
アメリカ合衆国カリフォルニア州94087,  
サニーヴェイル, ユーコン・ドライブ  
1477

**THIS PAGE BLANK (USPTO)**

**THIS PAGE BLANK (USPTO)**